EXHIBIT 1

US006823004B1

(12) **United States Patent**
Abdelilah et al.

(10) **Patent No.: US 6,823,004 B1**
(45) **Date of Patent: Nov. 23, 2004**

(54) **METHODS, SYSTEMS AND COMPUTER PROGRAM PRODUCTS FOR MONITORING PERFORMANCE OF A MODEM DURING A CONNECTION**

(75) Inventors: **Youssef Abdelilah**, Holly Springs, NC (US); **Gordon Taylor Davis**, Chapel Hill, NC (US); **Jeffrey Haskell Derby**, Chapel Hill, NC (US); **Ajay Dholakia**, Gattikon (CH); **Michelle Chan Granholm**, Morrisville, NC (US); **Dongming Hwang**, Cary, NC (US); **Fredy D. Neeser**, Langnau (CH); **Robert John Schule**, Cary, NC (US); **Malcolm Scott Ware**, Raleigh, NC (US); **Hua Ye**, Durham, NC (US); **Charles Robert Young**, Cary, NC (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/430,097**

(22) Filed: **Oct. 29, 1999**

(51) **Int. Cl.**[7] **H04B 1/38**; H04B 5/16

(52) **U.S. Cl.** **375/222**

(58) **Field of Search** 375/222, 219; 717/130; 714/712; 703/211; 710/20, 63; 379/3

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,557,308	A	1/1971	Alexander et al.	178/69.5
3,622,877	A	11/1971	MacDavid et al.	324/73 R
3,683,120	A	8/1972	Schenkel	179/15 A
3,729,717	A	4/1973	de Koe et al.	340/172.5
4,112,427	A	9/1978	Hofer et al.	340/347

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

EP	0 473 116 A2	8/1991	H04N/1/00
EP	0 659 007 A2	11/1994	H04M/11/06
EP	0 669 740 A2	12/1994	H04L/27/00
FR	2 345 019	3/1976	H04L/27/10
WO	WO 96/18261	6/1996	H04M/11/00
WO	WO 98/37657	8/1998	

OTHER PUBLICATIONS

Erup, et al., *Interpolation in Digital Modems—Part II: Implementation and Performance, IEEE Transactions on Communications*, vol. 41, No. 6, pp. 998–1008 (Jun. 1993).

Fischer, *Signal Mapping for PCM Modems, V–pcm Rapporteur Meeting*, Sunriver, Oregon, USA, , 5 pgs. (Sep. 4–12, 1997).

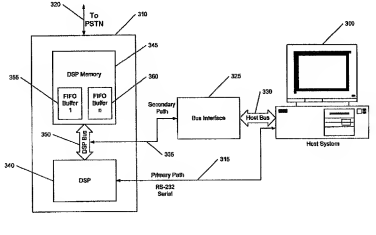
(List continued on next page.)

Primary Examiner—Stephen Chin
Assistant Examiner—Curtis Odom
(74) *Attorney, Agent, or Firm*—Scott W. Reid; Myers, Bigel, Sibley & Sojovec, P.A.

(57) **ABSTRACT**

Methods, systems and computer program products are provided for monitoring performance of a modem which obtain diagnostic data directly from memory associated with the modem's digital signal processor (DSP). A secondary path to the DSP memory is utilized for the monitoring operations so that real time data can be obtained during connection startup procedures and during an active connection. First-in first-out (FIFO) buffers are incorporated in the DSP memory to track state transitions of one or more of the state machines within the modem and various performance data measurements may be obtained directly from the DSP memory responsive to different state transition events. The real time collected data may be stored in a file and provided to a remote location for use in diagnosing customer problems with specific customer line connections. Accordingly, real time monitoring of digital and analog line conditions and modem performance may be utilized to diagnose problems with modems and line connections.

28 Claims, 4 Drawing Sheets



U.S. PATENT DOCUMENTS

4,132,242	A	1/1979	Carroll, Jr.	137/263
4,208,630	A	6/1980	Martinez	375/7
4,237,552	A	12/1980	Aikoh et al.	370/83
4,270,027	A	5/1981	Agrawal et al.	179/81 R
4,434,322	A	2/1984	Ferrell	178/22.13
4,450,556	A	5/1984	Boleda et al.	370/58
4,577,310	A	3/1986	Korsky et al.	370/58
4,578,796	A	3/1986	Charalambous et al.	375/8
4,720,861	A	1/1988	Bertrand	381/36
4,731,816	A	3/1988	Hughes-Hartogs	379/98
4,756,007	A	7/1988	Qureshi et al.	375/37
4,760,598	A	7/1988	Ferrell	380/44
4,797,898	A	1/1989	Martinez	375/7
4,833,706	A	5/1989	Hughes-Hartogs	379/98
4,868,863	A	9/1989	Hartley et al.	379/98
4,884,285	A	11/1989	Heynen et al.	375/25
4,890,303	A	12/1989	Bader	375/107
4,890,316	A	12/1989	Walsh et al.	379/98
4,894,847	A	1/1990	Tjahjadi et al.	375/121
4,901,333	A	2/1990	Hodgkiss	375/98
4,943,980	A	7/1990	Dobson et al.	375/42
4,953,210	A	8/1990	McGlynn et al.	380/48
4,967,413	A	10/1990	Otani	371/37.4
4,972,360	A	11/1990	Cuckier et al.	364/724.04
4,985,902	A	1/1991	Gurcan	375/14
4,991,169	A	2/1991	Davis et al.	370/77
4,995,030	A	2/1991	Helf	370/32.1
5,005,144	A	4/1991	Nakajima et al.	364/565
5,007,047	A	4/1991	Sridhar et al.	370/32.1
5,014,299	A	5/1991	Klupt et al.	379/98
5,033,062	A	7/1991	Morrow et al.	375/7
5,038,365	A	8/1991	Belloc et al.	375/8
5,040,190	A	8/1991	Smith et al.	375/4
5,052,000	A	9/1991	Wang et al.	371/43
5,058,134	A	10/1991	Chevillat et al.	375/39
5,065,410	A	11/1991	Yoshida et al.	375/98
5,067,125	A	11/1991	Tsuchida	370/79
5,068,875	A	11/1991	Quintin	375/78
5,107,520	A	4/1992	Karam et al.	375/60
5,111,481	A	5/1992	Chen et al.	375/14
5,119,401	A	6/1992	Tsujimoto	375/14
5,119,403	A	6/1992	Krishnan	375/39
5,134,611	A	7/1992	Steinka et al.	370/79
5,142,552	A	8/1992	Tzeng et al.	375/14
5,157,690	A	10/1992	Buttle	375/14
5,187,732	A	2/1993	Suzuki	379/5
5,210,755	A	5/1993	Nagler et al.	370/108
5,225,997	A	7/1993	Lederer et al.	364/550
5,253,272	A	10/1993	Jaeger et al.	375/60
5,253,291	A	10/1993	Naseer et al.	379/406
5,265,151	A	11/1993	Goldstein	379/97
5,285,474	A	2/1994	Chow et al.	375/13
5,291,479	A	3/1994	Vaziri et al.	370/58.2
5,311,578	A	5/1994	Bremer et al.	379/97
5,317,594	A	5/1994	Goldstein	375/8
5,351,134	A	9/1994	Yaguchi et al.	358/435
5,353,280	A	10/1994	Ungerböck	370/32.1
5,386,438	A	1/1995	England	375/121
5,394,110	A	2/1995	Mizoguchi	329/304
5,394,437	A	2/1995	Ayanoglu et al.	375/222
5,398,303	A	3/1995	Tanaka	395/51
5,402,445	A	3/1995	Matsuura	375/229
5,406,583	A	4/1995	Dagdeviren	375/5
5,418,842	A	5/1995	Cooper	379/98
5,432,794	A	7/1995	Yaguchi	371/5.5
5,434,884	A	7/1995	Rushing et al.	375/235
5,475,711	A	12/1995	Betts et al.	375/240
5,491,720	A	2/1996	Davis et al.	375/222
5,513,216	A	4/1996	Gadot et al.	375/233
5,519,703	A	5/1996	Chauffour et al.	370/84
5,528,625	A	6/1996	Ayanoglu et al.	375/222
5,528,679	A	6/1996	Taarud	379/34
5,533,048	A	7/1996	Dolan	375/222
5,534,913	A	7/1996	Majeti et al.	348/7
5,546,395	A	8/1996	Sharma et al.	370/84
5,563,908	A	10/1996	Kaku et al.	375/222
5,566,211	A	10/1996	Choi	375/332
5,598,401	A	1/1997	Blackwell et al.	379/94
5,625,643	A	4/1997	Kaku et al.	375/222
5,634,022	A	5/1997	Crouse et al.	395/704
5,640,387	A	6/1997	Takahashi et al.	370/359
5,646,958	A	7/1997	Tsujimoto	375/233
5,671,250	A	9/1997	Bremer et al.	375/222
5,694,420	A	12/1997	Ohki et al.	375/222
5,710,792	A	1/1998	Fukawa et al.	375/229
5,724,393	A	3/1998	Dagdeviren	375/296
5,726,765	A	3/1998	Yoshida et al.	358/412
5,729,226	A	3/1998	Betts et al.	341/94
5,732,104	A	3/1998	Brown et al.	375/222
5,734,663	A	3/1998	Eggenberger	371/39.1
5,751,717	A	5/1998	Babu et al.	370/466
5,751,796	A	5/1998	Scott et al.	379/93.31
5,754,594	A	5/1998	Betts et al.	375/285
5,757,849	A	5/1998	Gelblum et al.	375/222
5,757,865	A	5/1998	Kaku et al.	375/344
5,761,247	A	6/1998	Betts et al.	375/316
5,768,311	A	6/1998	Betts et al.	375/222
5,778,024	A	7/1998	McDonough	375/216
5,784,377	A	7/1998	Baydar et al.	370/463
5,784,405	A	7/1998	Betts et al.	375/222
5,784,415	A	7/1998	Chevillat et al.	375/341
5,793,809	A	8/1998	Holmquist	375/242
5,796,808	A	8/1998	Scott et al.	379/93.31
5,801,695	A	9/1998	Townshend	375/340
5,805,669	A	9/1998	Bingel et al.	379/28
5,809,075	A	9/1998	Townshend	375/254
5,812,537	A	9/1998	Betts et al.	370/286
5,815,534	A	9/1998	Glass	375/326
5,822,371	A	10/1998	Goldstein et al.	375/242
5,825,816	A	10/1998	Cole et al.	375/222
5,825,823	A	10/1998	Goldstein et al.	375/286
5,831,561	A	11/1998	Cai et al.	341/106
5,835,532	A	11/1998	Strolle et al.	375/233
5,835,538	A	11/1998	Townshend	375/295
5,838,724	A	11/1998	Cole et al.	375/222
5,839,053	A	11/1998	Bosch et al.	455/13.1
5,844,940	A	12/1998	Goodson et al.	375/222
5,850,388	A	12/1998	Anderson et al.	370/252
5,850,421	A	12/1998	Misra et al.	375/354
5,852,630	A *	12/1998	Langberg et al.	375/219
5,852,631	A	12/1998	Scott	375/222
5,862,141	A	1/1999	Trotter	370/468
5,862,179	A	1/1999	Goldstein et al.	375/242
5,862,184	A	1/1999	Goldstein et al.	375/295
5,870,429	A	2/1999	Moran, III et al.	375/222
5,872,817	A	2/1999	Wei	375/341
5,881,066	A	3/1999	Lepitre	371/20.5
5,881,102	A	3/1999	Samson	375/222
5,887,027	A	3/1999	Cohen et al.	375/222
5,911,115	A	6/1999	Nair et al.	455/63
5,914,982	A	6/1999	Bjarnason et al.	375/222
5,918,204	A	6/1999	Tsurumaru	704/214
5,926,506	A	7/1999	Berthold et al.	375/222
6,108,720	A *	8/2000	Tal et al.	370/282
6,272,452	B1 *	8/2001	Wu et al.	703/24
6,404,804	B1 *	6/2002	Mannering et al.	375/222
6,434,161	B1 *	8/2002	Higbee et al.	370/413

OTHER PUBLICATIONS

Gardner, *Interpolation in Digital Modems—Part I: Fundamentals, IEEE Transactions on Communications,* vol. 41, No. 3, pp. 501–507 (Mar. 1993).

Humblet et al., *The Information Driveway, IEEE Communications Magazine,* pp. 64–68 (Dec. 1996).

Kalet et al., *The Capacity of PCM Voiceband Channels, IEEE International Conference on Communications '93,* pp. 507–511 (Geneva, Switzerland, May 23–26, 1993).

Mueller et al., *Timing Recovery in Digital Synchronous Data Receiver, IEEE Transactions on Communications,* vol. Com–24, No. 5, pp. 516–531 (May 1976).

Okubo et al., *Building Block Design of Large Capacity PCM–TDMA Subscriber System and Direct Digital Interface to Digital Exchange,* Japan Radio Co., Ltd., pp. 69–73 (Japan).

Pahlavan et al., *Nonlinear Quantization and the Design of Coded and Uncoded Signal Constellations, IEEE Transactions on Communications,* vol. 39, No. 8, pp. 1207–1215 (Aug. 1991).

Proakis, *Digital Signaling Over a Channel with Intersymbol Interference, Digital Communications,* pp. 373, 381, (McGraw–Hill Book Company, 1983).

Williams et al., *Counteracting the Quantisation Noise from PCM Codecs,* BT Laboratories, pp. 24–29 (UK).

A Digital Modem and Analogue Modem Pair for Use on the Public Switched Telephone Network (PSTN) at Data Signalling Rates of Up to 56 000 Bit/s Downstream and 33 600 Bit/s Upstream, ITU–T V.90 (Sep. 1998).

Series V: Data Communication Over the Telephone Network: Interfaces and voiceband modems: A modem operating at data signalling rates of up to 33 600 bit/s for use on the general switched telephone network and on leased point–to–point 2–wire telephone type circuits, ITU–T V.34 (10/96).

Bell, R.A., et al., *Automatic Speed Reduction and Switched Network Back–up, IBM Technical Disclosure Bulletin,* vol. 32, No. 1, pp. 154–157 (Jun. 1989).

Abbiate, J.C., et al, *Variable–Data Transmission Modem, IBM Technical Disclosure Bulletin,* vol. 17, No. 11, pp. 3301–3302 (Apr. 1975).

Data Communication Over the Telephone Network: Procedures for Starting Sessions of Data Transmission Over the General Switched Telephone Network, ITU–T V.8 (09/94).

Line Quality Monitoring Method, IBM Technical Disclosure Bulletin, vol. 18, No. 8, pp. 2726–2726 (Jan. 1976).

Loopback Tests for V.54 Data Communication Equipment, IBM Technical Dislosure Bulletin, vol. 32, No. 3A, pp. 295–299 (Aug. 1989).

On–Line Real Time Modem Testing, IBM Technical Disclosure Bulletin, vol. 20, No. 6, pp. 2252–2254 (Nov. 1977).

Pierobon, Gianfranco L., *Codes of Zero Spectral Density at Zero Frequency, IEEE Transactions on Information Theory,* vol. IT–30, No. 2, pp. 435–439 (Mar., 1984).

Marcus, Brian H, et al., *On Codes with Spectral Nulls at Rational Submultiples of the Symbol Frequency, IEEE Transactions on Information Theory,* vol. IT–33, No. 4, pp. 557–568 (Jul. 1987).

Fischer, Robert, et al., *Signal Mapping for PCM Modems, ITU–Telecommunications Standardization Sector PCM '97–120, Vpcm Rapporteur Meeting,* (Sunriver, Oregon: Sep. 4–12, 1997).

Pulse Code Modulation (PCM) of Voice Frequencies, ITU–T, Recommendation G.711 (Geneva, 1972).

Data Communication Over the Telephone Network; Error–Correcting Procedures for DCEs Using Asynchronous–to–Synchronous Conversion, ITU–T–V.42 (03/93).

Improvement to Spectral Shaping Technique, Research Disclosure, v. 41, n415, 415111, pp. 1550–1551 (Nov. 1998).

TIA Standard Draft: North American Telephone Network Transmission Model for Evaluating Analog Client to Digitally Connected Server Modems, Telecommunications Industry Association, PN3857, Draft 10 (Feb. 1999).

Davis, Gordon T., *DSP and MATLAB implementation of model–based constellation generation* (Sep. 18, 1998).

Woodruff, K.R, et al, *Automatic and Adaptive System and Efficient Communication in Noisy Communication Line Environments, IBM Technical Disclosure Bulletin,* vol. 24, No. 9, pp. 4627–4629 (Feb. 1982).

Godard, D., et al., *Decision Feedback Equalizer Stabilization in Adaptive Mode, IBM Technical Disclosure Bulletin,* vol. 24, No. 11A, pp. 5691–5692 (Apr. 1982).

Borgnis–Desbordes, P., et al., *Variable–Speed Data Transmission, IBM Technical Disclosure Bulletin,* vol. 27, No. 4A, pp. 2269–2270 (Sep. 1984).

Couland, G., et al., *Analog Wrap Self–Test in Modems During Retrain Operations, IBM Technical Disclosure Bulletin,* vol. 28, No. 6, p. 2457 (Nov. 1985).

Maddens, F., *Sixteen–State Forward Convolutional Encoder, IBM Technical Disclosure Bulletin,* vol. 28, No. 6, pp. 2466–2468 (Nov. 1985).

Remote Modem–Type Self–Learning, IBM Technical Disclosure Bulletin, vol 28, No. 6, pp. 2398–2399 (Nov. 1985).

Maddens, F., *Sixteen–State Feedback Convolutional Encoder, IBM Technical Disclosure Bulletin,* vol. 28, No. 10, pp. 4212–4213 (Mar. 1986).

Nobakht, R.A., *Trellis–Coded Modulation Coding Scheme for a 19.2 Kbps Modem, IBM Technical Disclosure Bulletin,* vol. 36, No. 11, pp. 167–170 (Nov. 1993).

Nobakht, R.A., *Unified Table Based Subset Decoder for the Viterbi Algorithm, IBM Technical Disclosure Bulletin,* vol. 37, No. 9, pp. 581–587 (Sep. 1994).

Nobakht, R.A., *Trellis Subset Decoder Algorithm Based on a Pattern Recognition Scheme, IBM Technical Disclosure Bulletin,* vol. 37, No. 10, pp. 693–697 (Oct. 1994).

Barlet, J., et al., *Full Speed Recovery in High Speed Modems, IBM Technical Disclosure Bulletin,* vol. 23, No. 2, pp. 641–643 (Jul. 1980).

Dialog Abstract, *Sample rate converter for duplex modem,* European Patent No. 285413.

Dialog Abstract, *Two–speed full–duplex modem for telephone network,* PCT No. WO8501407.

Dialog Abstract, *Digital data transmission system,* European Patent No. 124674.

Dialog Abstract, *Facsimile communication controller,* Japanese Publication No. 04–175060 (Jun. 23, 1992).

Dialog Abstract, *Picture communication equipment,* Japanese Publication No. 03–120954 (May 23, 1991).

Dialog Abstract, *Radio date transmission system,* Japanese Publication No. 01–179535 (Jul. 17 ,1989).

Dialog Abstract, *Facsimile device,* Japanese Publication No. 57–164654 (Oct. 9, 1982).

Dialog Abstract, *Data repeater,* Japanese Publication No. 57–087255 (May 31, 1982).

Dialog Abstract, *Blinding training method for decision feedback equaliser having feed–forward and feedback filters,* European Patent No. 880253.

Dialog Abstract, *Processing method for distorted signal received by qam receiver,* European Patent No. 465851.

Dialog Abstract, *Establishing wireless communication channel,* PCT No. WO 9905820.

Dialog Abstract, *High–speed rate adaptive subscriber line digital data modem,* PCT No. WO 9830001.

Dialog Abstract, *Digital modem in digital modulation system,* Japanese Patent No. 8116341.

Dialog Abstract, *Communication equipment and radio communication adapter,* Japanese Publication No. 08–340289 (Dec. 24, 1996).

Dialog Abstract, *Data recording method,* Japanese Publication No. 05–089597 (Apr. 9, 1993).

Dialog Abstract, *Transmission control system for data communication and its modem equipment,* Japanese Publication No. 02–228853 (Sep. 11, 1990).

Naguib, A.F., et al., Dialog Abstract, *A space–time coding modem for high–data–rate wireless communications, IEEE Journal of Selected Areas in Communications,* vol. 16, No. 8, pp. 1459–1478 (Oct. 1998).

Denno, S., et al., Dialog Abstract, *Mbit/s burst modem with an adaptive equalizer for TDMA mobile radio communications, IEICE Transactions on Communications,* vol. E81–B, No. 7, pp. 1453–1461 (Jul. 1998).

Naguib, A.F., et al., Dialog Abstract, *A space–time coding modem for high–data–rate wireless communications, GLOBECOM 97, IEEE Global Telecommunications Conference,* vol. 1, pp. 102–109 (1997).

Kobayashi, K., et al., Dialog Abstract, *Fully digital burst modem for satellite multimedia communication systems, IEICE Transactions on Communications,* vol. E80–B, No. 1, pp. 8–15 (Jan. 1997).

Skellern, D.J., et al., Dialog Abstract, *A high speed wireless LAN, IEEE Micro,* vol. 17, No. 1, pp. 40–47 (Jan.–Feb. 1997).

Enomoto, K. et al., Dialog Abstract, *A mode switching type burst demodulator AFC, Transactions of the Institute of Electronics, Information and Communication Engineers,* vol. J76B–II, No. 5, pp. 415–421 (May 1993).

Betts, W., Dialog Abstract, *Nonlinear encoding by surface projection, International Conference on Data Transmission—Advances in Modem and ISDN Technology and Applications* (Sep. 23–25, 1992).

Schilling, D.L., et al., Dialog Abstract, *The FAVR meteor burst communication experiment, Military Communications in a Changing World MILCOM '91* (Nov. 4–7, 1991).

Jacobsmeyer, J.M., Dialog Abstract, *Adaptive trellis–coded modulation for bandlimited meteor burst channels, IEEE Journal on Selected Areas in Communications,* vol. 10, No. 3, pp. 550–561 (Apr. 1992).

Sato, T., et al., Dialog Abtsract, *Protocol configuration and verification of an adaptive error control scheme over analog cellular networks, IEEE Transactions on Vehicular Technology,* vol. 41, No. 1, pp. 69–76 (Feb. 1992).

Lee, L.–N., et al., Dialog Abstract, *Digital signal processor–based programmable BPSK/QPSK/offset–QPSK modems, COMSAT Technical Review,* pp .195–234 (Fall 1989).

Sato, T., et al., Dialog Abstract, *Error–free high–speed data modem, Oki Technical Review,* vol. 56, No. 133, pp. 20–26 (Apr. 1989).

Seo, J.–S. et al., Dialog Abstract, *Performance of convolutional coded SQAM in hardlimited satellite channels, IEEE International Conference on Communications BOSTON–ICC/89,* vol. 2, pp. 787–791 (Jun. 11–14, 1989).

Murakama, K., et al., Dialog Abstract, *FEC combined burst–modem for business satellite communications use, IEEE/IECE Global Telecommunications Conference 1987,* vol. 1, pp. 274–280 (Japan, Nov. 15–18, 1987).

McVerry, F., Dialog Abstract, *Performance of a fast carrier recovery scheme for burst–format DQPSK transmission over satellite channels, International Conference on Digital Processing of Signals in Communications,* pp. 165–172 (United Kingdom, 1985).

Filter, J.H.J., Dialog Abstract, *An algorithm for detecting loss of synchronisation in data transmission test sets (modems), Transactions of the South African Institute of Electrical Engineers,* vol. 76, No. 1, pp. 39–43 (Jan. 1985).

Gersho, A., Dialog Abstract, *Reduced compexity implementation of passband adaptive equlizers, IEEE Journal on Selected Areas in Communications,* vol. SAC–2, No. 5, pp. 778–779 (Sep. 1984).

Dialog Abstract, *High–speed full–duplex modem reduces telephone connect line, EDN,* vol. 27, No. 18, p. 77 (Sep. 15, 1982).

Chadwick, H., et al., Dialog Abstract, *Performance of a TDMA burst modem through a dual nonlinear satellite channel, Fifth Internaitonal Conference on Digital Satellite Communications,* pp. 63–67 (Italy, Mar. 23–26, 1981).

Nussbaumer, H., Dialog Abstract, *Reducing the acqusition time in an automatic equalizer, IBM Technical Disclosure Bulletin,* vol. 18, No. 5, pp. 1465–1479 (Oct. 1975).

Uzunoglu, V., et al., Dialog Abstract, *Synchronous and the coherent phase–locked synchronous oscillators: new techniques in synchronization and tracking, IEEE Transactions on Circuits and Systems,* vol. 36, No. 7, pp. 997–1004 (Jul. 1989).

Minei, I., et al., Dialog Abstract, *High–speed Internet access through unidirectional geostationary satellite channels, IEEE Journal on Selected Areas in Communications,* vol. 17, No. 2, pp. 345–359 (Feb. 1999).

Ovadia, S., Dialog Abstract, *The effect of interleaver depth and QAM channel frequency offset on the performance of multichannel AM–VSB/256–QAM video lightwave transmission systems, International Conference on Telecommunications: Bridging East and West Through Communications,* vol. 1, pp. 339–343 (Greece, Jun. 21–25, 1998).

Johnson, R.W., et al., Dialog Abstract, *Error correction coding for serial–tone HG transmission, Seventh International Conference on HF Radio Systems and Techniques,* pp. 80–84 (United Kingdom, Jul. 7–10, 1997).

Karasawa, Y., et al., Dialog Abstract, *Cycle slip in clock recovery on frequency–selective fading channels, IEEE Transactions on Communications,* vol. 45, No. 3, pp. 376–383 (Mar. 1997).

Umehira, M., et al., Dialog Abstract, *Design and performance of burst carrier recovery using a phase compensated filter, Transactions of the Institute of Electronics, Information and Communication Engineers,* vol. J78B–II, No. 12, pp. 735–746 (Dec. 1995).

De Bot, P., et al., Dialog Abstract, *An example of a multi–resolution digital terrestrial TV modem, Proceedings of ICC '93—IEEE International Conference on Communications,* vol. 3, pp. 1785–1790 (Switzerland, May 23–26, 1993).

Lei, Chen, et al., Dialog Abstract, *Single–tone HF high speed data modem, Proceedings of TENCON '93—IEEE Region 10 International Conference on Computers, Communications and Automation*, vol. 3, pp. 94–98 (China, Oct. 19–21, 1993).

Werner, B.D., et al., Dialog Abstract, *Simulation issues for future wireless modems, IEEE Communications*, vol. 32, No. 7, pp. 42–53 (Jul. 1994).

Sato, T., et al., Dialog Abstract, *Vehicle terminal equipment with dedicated DSP, Oki Technical Review*, vol. 58, No. 144, pp. 49–52 (Jul. 1992).

Sato, T., et al., Dialog Abstract, *Protocol configuration and verification of an adaptive error control scheme over analog cellular networks, IEEE Transactions on Vehicular Technology*, vol. 41, No. 1, pp. 69–76 (Feb. 1992).

Tamm, Yu. A., Dialog Abstract, *The effect of suppressing harmonic interference using an adaptive equalizer, Elektrosvyaz*, vol. 45, No. 3, pp. 5–10 (Mach 1990).

Saleh, A.A.M., et al., Dialog Abstract, *An experimental TDMA indoor radio communications systemusing slow frequency hopping and coding, IEEE Transactions on Communications*, vol. 39, No. 1, pp. 152–162 (Jan., 1991).

Nergis, A., Dialog Abstract, *Optimum HF digital communication systems with block coding and interleaving techniques, Proceedings of the 1990 Bilkent International Conference on New Trends in Communication, Control and Signal Processing*, vol. 1, pp. 511–517 (Turkey, Jul. 2–5, 1990).

Kawamata, F., et al., Dialog Abstract, *An evaluation of voice codecs and facsimiles, Review of the Communicatons Research Laboratory*, vol. 36, pp. 69–73 (Mar. 1990).

Sato, T., et al., Dialog Abstract, *Error–free high–speed data transmission protocol simultaneously applicable to both wire and mobile radio channels, 38th IEEE Vehicular Technology Conference: 'Telecommunications Freedom—Technology on the Move'*, pp. 489–496 (Jun. 15–17, 1988).

Dialog Abstract, *1200–bit/s cellular modem DLD03H, Oki Technical Review*, vol. 53, No. 127, pp. 70–72 (Jul. 1987).

Chamberlin, J.W., et al., Dialog Abstract, *Design and field test of a 256–QAM DIV modem, IEEE Journal on Selected Areas in Communications*, vol. SAC–5, No. 3, pp. 349–356 (Apr. 1987).

De Cristofaro, R., et al., Dialog Abstract, *A 120 Bv/s QPSK modem designed for the INTELSAT TDMA network, International Journal of Satellite Communications*, vol. 3, Nos. 1–2, pp. 145–160 (Jan.Jun. 1985).

Shumate, A., Dialog Abstract, *Error correction coding for channels subject to occasional losses of bit count integrity, IEEE Military Communications Conference*, vol. 1, pp .89–83 (Oct. 21–24, 1984).

Suyderhoud, H., et al., Dialog Abstract, *Investigation of 9.6 kb/s data transmission via a PCM link at 64 kb/s with and without link errors, International Journal of Satellite Communications*, vol. 2, No. 1, pp. 81–87 (Jan.–Mar. 1984).

Smith, C., Dialog Abstract, *Relating the performance of speech processors to the bit error rate, Speech Technology*, vol. 2, No. 1, pp. 41–53 (Sep.–Oct. 1983).

Suyderhoud, H., et al., Dialog Abstract, *Investigation of 9.6–kbit/s data transmission via a PCM link at 64 kbit/s with and without link errors, Sixth International Conference on Digital Satellite Communications Proceedings*, pp. 26–33 (Sep. 19, 23, 1983).

Kittel, L., Dialog Abstract, *Analogue and discrete channel models for signal transmission in mobile radio, Frequenz*, vol. 36, Nos. 4–5, pp. 153–160 (Apr.–May 1982).

Farrell, P.G., et al., Dialog Abstract, *Soft–decision error control of h.f. data transmission, IEE Proceedings F (Communications, Radar and Signal Processing)*, vol. 127, No. 5, pp. 389–400 (Oct. 1980).

Johnson, A.L., Dialog Abstract, *Simulation and implementation of a modulation system for overcoming ionospheric scintillation fading, AGARD Conference Proceedings No. 173 on Radio Systems and the Ionosphere*, pp. 3/1–5 (Greece, May 26–30, 1975).

Matsumura, K., et al., Dialog Abstract, *Anti–interference data–transmission set of HF radio equipment, Mitsubishi Electric Engineer*, No. 41, pp. 18–23 (Sep., 1974).

Blank, H.A., et al., Dialog Abstract, *A Markov error channel model, 1973 National Telecommunications Conference*, vol. 1, pp. 15B/1–8 (Nov. 26–28, 1973).

McGruther, W.G. Dialog Abstract, *Long term error performance data for operation at 2400 bps ona nonswitched private line network, Summaries of papers presented at 1970 Canadian symposium on communications*, pp. 65–66 (Canada, Nov. 12–13, 1970).

Burton, H.O., et al., Dialog Abstract, *On the use of error statistics from data transmission on telephone facilities to estimate performance of forward–error–correction, 1970 international conference on communications*, p. 21 (Jun. 8–10, 1970).

Bowen, R.R., Dialog Abstract, *Applications on burst error correction codes to data modems for dispersive channels, Proceedings of the 1970 international symposium on information theory*, p. 1 (Netherlands, Jun. 15–19, 1970).

Pierce, A.W., et al., Dialog Abstract, *Effective application of forward–acting error–control coding to multichannel h.f. data modems, IEEE Transactions on Communication Technology*, vol. Com–18, No. 4, pp. 281–294 (Aug. 1970).

West, R.L., Abstract, *Data Concentration Method, IBM Technical Disclosure Bulletin*, pp. 487–489; http://w3.infogate.ibm.com:1207/SESS506884/GETDOC/39/2/1 (Jul., 1975).

Haas, L.C., et al., Abstract, *Received Line Signal Quality Analysis, IBM Technical Disclosure Bulletin*, pp. 5414–5416: http://w3.infogate.ibm.com:1207/SESS506884/GETDOC/43/1/1 (May, 1981).

Nussbaumer, H., Abstract, *Reducing the Acquisition Time in an Automatic Equalizer, IBM Technical Disclosure Bulletin*, pp. 1465–1479; http://w3.infogate.ibm.com:1207/SESS506884/GETDOC/40/2/1 (Oct. 1975).

Dialog Abstract, *Listener echo canceller for digital communication system*, PCT No. WO 9310607.

Dialog Abstract, *Reduced time remote access method for modem computer*, PCT No. WO 9209165.

* cited by examiner

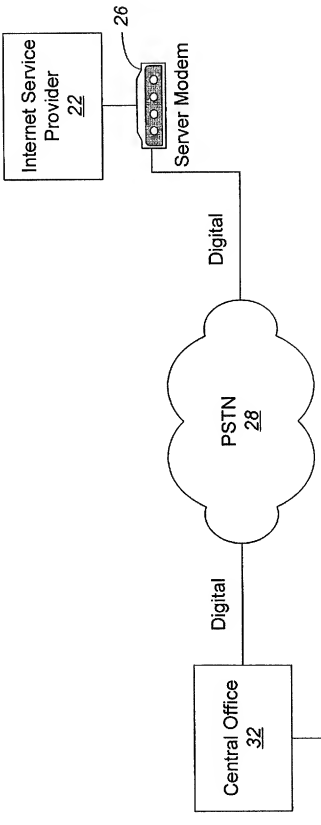


FIG. 1
(PRIOR ART)

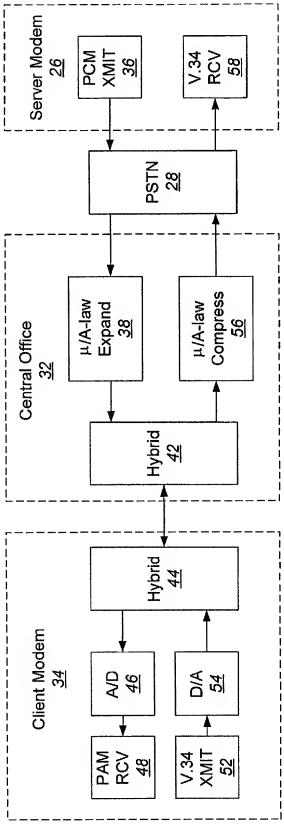


FIG. 2
(PRIOR ART)

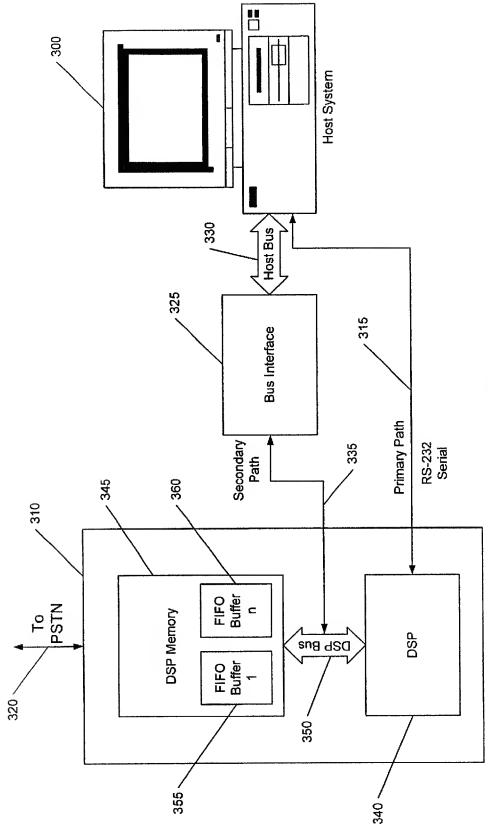

300
Host System
330
Host Bus
325
Bus Interface
315
Secondary Path
335
Primary Path
RS-232 Serial
310
345
360
To PSTN
320
DSP Memory
FIFO Buffer n
FIFO Buffer 1
DSP Bus
DSP
355
350
340

FIG. 3

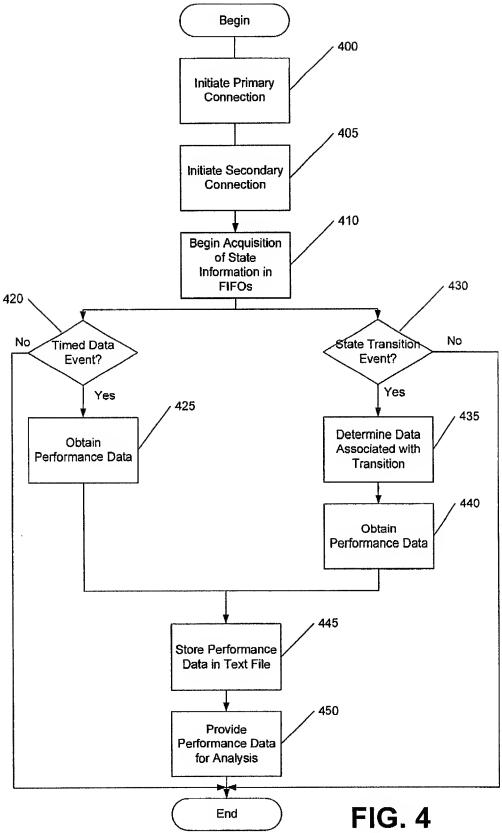

Begin
400
Initiate Primary Connection
405
Initiate Secondary Connection
410
Begin Acquisition of State Information in FIFOs
420
No
Timed Data Event?
Yes
425
Obtain Performance Data
430
State Transition Event?
No
Yes
435
Determine Data Associated with Transition
440
Obtain Performance Data
445
Store Performance Data in Text File
450
Provide Performance Data for Analysis
End

FIG. 4

METHODS, SYSTEMS AND COMPUTER PROGRAM PRODUCTS FOR MONITORING PERFORMANCE OF A MODEM DURING A CONNECTION

FIELD OF THE INVENTION

The present invention relates generally to the field of modems, and, more particularly, to modem diagnostics.

BACKGROUND OF THE INVENTION

The demand for remote access to information sources and data retrieval, as evidenced by the success of services such as the World Wide Web, is a driving force for high-speed network access technologies. Today's telephone network offers standard voice services over a 4 kHz bandwidth. Traditional analog modem standards generally assume that both ends of a modem communication session have an analog connection to the public switched telephone network (PSTN). Because data signals are typically converted from digital to analog when transmitted towards the PSTN and then from analog to digital when received from the PSTN, data rates may be limited to 33.6 kbps as defined in the V.34 transmission recommendation developed by the International Telecommunications Union (ITU).

The need for an analog modem can be eliminated, however, by using the basic rate interface (BRI) of the Integrated Services Digital Network (ISDN). A BRI offers end-to-end digital connectivity at an aggregate data rate of 160 kbps, which is comprised of two 64 kbps B channels, a 16 kbps D channel, and a separate maintenance channel. The ISDN offers suitable data rates for Internet access, telecommuting, remote education services, and some forms of video conferencing. ISDN deployment, however, has been very slow due to the substantial investment required of network providers for new equipment. Because the ISDN is not very pervasive in the PSTN, the network providers have typically tarriffed ISDN services at relatively high rates, which may be ultimately passed on to the ISDN subscribers. In addition to the high service costs, subscribers must generally purchase or lease network termination equipment to access the ISDN.

While most subscribers do not enjoy end-to-end digital connectivity through the PSTN, the PSTN is nevertheless mostly digital. Typically, the only analog portion of the PSTN is the phone line or local loop that connects a subscriber or client modem (e.g., an individual subscriber in a home, office, or hotel) to the telephone company's central office (CO). In recent years, local telephone companies have been replacing portions of their original analog networks with digital switching equipment. Nevertheless, the connection between the home and the CO has been the slowest to change to digital as discussed in the foregoing with respect to ISDN BRI service. A recent data transmission recommendation issued by the ITU, known as V.90, takes advantage of the digital conversions that have been made in the PSTN. By viewing the PSTN as a digital network, V.90 technology is able to accelerate data downstream from the Internet or other information source to a subscriber's computer at data rates of up to 56 kbps, even when the subscriber is connected to the PSTN via an analog local loop.

To understand how the V.90 recommendation achieves this higher data rate, it may be helpful to briefly review the operation of V.34 analog modems. V.34 modems are optimized for the situation where both ends of a communication session are connected to the PSTN by analog lines. Even

though most of the PSTN is digital, V.34 modems treat the network as if it were entirely analog. Moreover, the V.34 recommendation assumes that both ends of the communication session suffer impairment due to quantization noise introduced by analog-to-digital converters. That is, the analog signals transmitted from the V.34 modems are sampled at 8000 times per second by a codec upon reaching the PSTN with each sample being represented or quantized by an eight-bit pulse code modulation (PCM) codeword. The codec uses 256, non-uniformly spaced, PCM quantization levels defined according to either the μ-law or A-law companding standard.

Because the analog waveforms are continuous and the binary PCM codewords are discrete, the digits that are sent across the PSTN can only approximate the original analog waveform. The difference between the original analog waveform and the reconstructed quantized waveform is called quantization noise, which limits the modem data rate.

While quantization noise may limit a V.34 communication session to 33.6 kbps, it nevertheless affects only analog-to-digital conversions. The V.90 standard relies on the lack of analog-to-digital conversions outside of the conversion made at the subscriber's modem to enable transmission at 56 kbps.

The general environment for which the V.90 standard was developed is depicted in FIG. **1**. An Internet Service Provider (ISP) **22** is connected to a subscriber's computer **24** via a V.90 digital server modem **26**, through the PSTN **28** via digital trunks (e.g., T1, E1, or ISDN Primary Rate Interface (PRI) connections), through a central office switch **32**, and finally through an analog loop to the client's modem **34**. The central office switch **32** is drawn outside of the PSTN **28** to better illustrate the connection of the subscriber's computer **24** and modem **34** into the PSTN **28**. It should be understood that the central office **32** is, in fact, a part of the PSTN **28**. The operation of a communication session between the subscriber **24** and an ISP **22** is best described with reference to the more detailed block diagram of FIG. **2**.

Transmission from the server modem **26** to the client modem **34** will be described first. The information to be transmitted is first encoded using only the 256 PCM codewords used by the digital switching and transmission equipment in the PSTN **28**. The PCM codewords are modulated using a technique known as pulse amplitude modulation (PAM) in which discrete analog voltage levels are used to represent each of the 256 PCM codewords. These PAM signals are transmitted towards the PSTN by the PAM transmitter **36** where they are received by a network codec. No information is lost in converting the PAM signals back to PCM because the codec is designed to interpret the various voltage levels as corresponding to particular PCM codewords without sampling the PAM signals. The PCM data is then transmitted through the PSTN **28** until reaching the central office **32** to which the client modem **34** is connected. Before transmitting the PCM data to the client modem **34**, the data is converted from its current form as either μ-law or A-law companded PCM codewords to PAM voltages by the codec expander (digital-to-analog (D/A) converter) **38**. These PAM voltages are processed by a central office hybrid **42** where the unidirectional signal received from the codec expander **38** is transmitted towards the client modem **34** as part of a bidirectional signal. A second hybrid **44** at the subscriber's analog telephone connection converts the bidirectional signal back into a pair of unidirectional signals. Finally, the analog signal from the hybrid **44** is converted into digital PAM samples by an analog-to-digital (A/D) converter **46**, which are received and

decoded by the PAM receiver **48**. Note that for transmission to succeed effectively at 56 kbps, there must be only a single digital-to-analog conversion and subsequent analog-to-digital conversion between the server modem **26** and the client modem **34**. Recall that analog-to-digital conversions in the PSTN **28** can introduce quantization noise, which may limit the data rate as discussed hereinbefore. Moreover, the PAM receiver **48** needs to be in synchronization with the 8 kHz network clock to properly decode the digital PAM samples.

Transmission from the client modem **34** to the server modem **26** follows the V.34 data transmission standard. That is, the client modem **34** includes a V.34 transmitter **52** and a D/A converter **54** that encode and modulate the digital data to be sent using techniques such as quadrature amplitude modulation (QAM). The hybrid **44** converts the unidirectional signal from the digital-to-analog converter **54** into a bidirectional signal that is transmitted to the central office **32**. Once the signal is received at the central office **32**, the central office hybrid **42** converts the bidirectional signal into a unidirectional signal that is provided to the central office codec. This unidirectional, analog signal is converted into either μ-law or A-law companded PCM codewords by the codec compressor (A/D converter) **56**, which are then transmitted through the PSTN **28** until reaching the server modem **26**. The server modem **26** includes a conventional V.34 receiver **58** for demodulating and decoding the data sent by the V.34 transmitter **52** in the client modem **34**. Thus, data is transferred from the client modem **34** to the server modem **26** at data rates of up to 33.6 kbps as provided for in the V.34 standard.

The V.90 standard only offers increased data rates (e.g., data rates up to 56 kbps) in the downstream direction from a server to a subscriber or client. Upstream communication still takes place at conventional data rates as provided for in the V.34 standard. Nevertheless, this asymmetry is particularly well suited for Internet access. For example, when accessing the Internet, high bandwidth is most useful when downloading large text, video, and audio files to a subscriber's computer. Using V.90, these data transfers can be made at up to 56 kbps. On the other hand, traffic flow from the subscriber to an ISP consists of mainly keystroke and mouse commands, which are readily handled by the conventional rates provided by the V.34 standard.

The V.90 standard, therefore, provides a framework for transmitting data at rates up to 56 kbps provided the network is capable of supporting the higher rates. The most notable requirement is that there can be at most one digital-to-analog conversion in the path from the server modem to the client modem. Nevertheless, other digital impairments, such as robbed bit signaling (RBS) and digital mapping through PADs, which results in attenuated signals, can also inhibit transmission at V.90 rates. Communication channels exhibiting non-linear frequency response characteristics are yet another impediment to transmission at the V.90 rates. These factors may limit conventional V.90 performance to less than the 56 kbps theoretical data rate.

Articles such as Humblet et al., "The Information Driveway," IEEE Communications Magazine, December 1996, pp. 64–68, Kalet et al., "The Capacity of PCM Voiceband Channels," IEEE International Conference on Communications '93, May 23–26, 1993, Geneva, Switzerland, pp. 507–511, Fischer et al., "Signal Mapping for PCM Modems," V-pcm Rapporteur Meeting, Sunriver, Oreg., USA, Sep. 4–12, 1997, and Proakis, "Digital Signaling Over a Channel with Intersymbol Interference," Digital Communications, McGraw-Hill Book Company, 1983, pp. 373, 381, provide general background information on digital communication systems.

As modem performance specifications push ever closer to the limits supported by the physical media, the potential for user dissatisfaction with modem performance under various line conditions increases. For example, it is not uncommon for 56 k modem users to experience significantly lower connection speeds or throughput than they may expect in light of the 56 k capability associated with the modems. Users may not appreciate that, while manufacturers of such modems often quote 53.3 kbps as a maximum speed, they also typically note that line conditions may result in even lower speeds. Unfortunately, manufacturers currently have generally not effectively explained why a given customer's modem may be operating at a particular speed or throughput. Furthermore, manufacturers are not readily able to explain why a given customer may encounter an inability to make a connection or experience dropped connections.

One known approach to evaluating modem performance is the use of AT commands, such as those provided for by operating systems, such as Windows™ from Microsoft Corporation, for communicating with a modem (such as the #UD command). However, only a limited amount of diagnostic information may be obtained from a modem using this approach. Furthermore, the modem communication session typically must be terminated to obtain information using AT commands, which not only interrupts ongoing operations but further may limit the amount and types of data available from the modem, (for example, due to retraining procedures overwriting various data within the modem). For example, an interface like Hyperterm™ may be used to enter AT commands which would preclude the use of such commands while the modem was in use by a user application, such as Dial-up Networking, as the Hyperterm™ application would establish any client modem to server modem connection. A further approach, as described in U.S. Pat. No. 5,634,022 entitled "Multi-Media Computer Diagnostic System" provides for insertion of branch instructions into the operating code of a signal processing device transferring operations to a diagnostic program. However, this approach is directed to discovery and correction of algorithmic or logical faults and may not be well suited to line condition monitoring. Accordingly, a need exists to obtain knowledge of customer specific line conditions and potential interworking problems between a customer's client modem and a server modem to address customer support problems.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide methods, systems and computer program products for monitoring performance of a modem which may be able to obtain data in real time.

It is a further object of the present invention to provide such methods, systems and computer program products which may provide obtained performance data to a provider for use in addressing customer support problems.

These and other objects, advantages, and features of the present invention may be provided by methods, systems and computer program products for monitoring performance of a modem which obtain diagnostic data directly from a memory associated with the modem's digital signal processor (DSP). A secondary path to the DSP memory is utilized for the monitoring operations so that real time data can be obtained during connection procedures and during an active connection. First-in first-out (FIFO) buffers are preferably

incorporated in the DSP memory to track state transitions of one or more of the state machines within the modem and various performance data measurements may be obtained directly from the DSP memory responsive to different state transition events. The real time collected data may be stored in a file and provided to a remote location for use in diagnosing customer problems with specific customer line connections. Accordingly, real time monitoring of digital and analog line conditions and modem performance may be utilized to diagnose problems with modems and line connections.

In one embodiment of the present invention, a method is provided for monitoring performance of a modem including obtaining data related to the performance of the modem from a memory of the modem during an active connection to the modem asynchronously with communication operations of the modem supporting the active connection. Preferably, the data related to the performance of the modem is obtained from a digital signal processor (DSP) memory using a secondary path to the DSP memory. The data may be obtained during startup of the active connection. The data further may be selected from the group consisting of line condition data and interworking data.

In a further embodiment of the present invention, the data related to the performance of the modem includes internal state information and obtaining operations include determining that a state transition has occurred based on the internal state information and capturing a selected type of data related to the performance of the modem responsive to a state transition. Furthermore, the modem may include a plurality of state machines each state machine having a plurality of associated states in which case determining operations may further include determining that a state transition of one of the plurality of state machines has occurred based on detecting a change from one of the plurality of associated states of the one of the plurality of state machines to another of the plurality of associated states of the one of the plurality of state machines and the selected type of data may be selected based on the one of the plurality of state machines. The selected type of data may further be selected based on the one of the plurality of associated states and the another of the plurality of associated states.

In another embodiment of the present invention, a first-in first-out (FIFO) buffer is provided in the DSP memory and determining operations further include placing the internal state information in the FIFO buffer. A plurality of FIFO buffers may be provided in the DSP memory, each of the plurality of FIFO buffers being associated with a different state machine of the modem and internal state information associated with a respective state machine of the modem may be placed in a corresponding one of the plurality of FIFO buffers.

In a further embodiment of the present invention, the date related to the performance of the modem is stored in a text file. The stored data may then be provided to a remote location for analysis.

In another aspect of the present invention, a system is provided for monitoring performance of a modem. The system includes a host system having a host system bus. The monitoring system further includes a modem. The modem includes a digital signal processor (DSP) and a memory coupled to the DSP over a DSP system bus. In addition, the modem also includes a primary path to the DSP memory that supports a communication connection. A bus interface couples the host system bus to the DSP system bus, the bus interface being configured to allow the host system to access the DSP memory to obtain data related to performance of the modem during an active communication session supported by the primary path of the modem asynchronously with communication operations of the modem supporting the active communication session. In one embodiment, the modem further includes a first-in first-out (FIFO) buffer coupled to the DSP and the DSP is configured to place internal state information associated with state machines of the modem in the FIFO buffer. Furthermore, the modem may include a plurality of first-in first-out (FIFO) buffers, each of the FIFO buffers being associated with one of a plurality of state machines of the modem and the DSP may be configured to place internal state information associated with the plurality of state machines in a corresponding one of the FIFO buffers.

As will further be appreciated by those of skill in the art, while described above primarily with reference to method aspects, the present invention may be embodied as methods, apparatus/systems and/or computer program products.

BRIEF DESCRIPTION OF THE DRAWINGS

Other features of the present invention will be more readily understood from the following detailed description of specific embodiments thereof when read in conjunction with the accompanying drawings, in which:

FIG. **1** is block diagram illustrating a typical V.90 connection between a subscriber and an ISP in accordance with the prior art;

FIG. **2** is a detailed block diagram of the internal architecture and connections between the client modem, the central office, and the server modem of FIG. **1**;

FIG. **3** is a block diagram of a modem performance monitoring system in accordance with the present invention; and

FIG. **4** is a flow chart illustrating operations for obtaining performance data according to an embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention will now be described more fully hereinafter with reference to the accompanying drawings, in which preferred embodiments of the invention are shown. This invention may, however, be embodied in different forms and should not be construed as limited to the embodiments set forth herein. Rather, these embodiments are provided so that this disclosure will be thorough and complete, and will fully convey the scope of the invention to those skilled in the art. Like reference numbers signify like elements throughout the description of the figures.

As will be appreciated by those skilled in the art, the present invention can be embodied as a method, a system, or a computer program product. Accordingly, the present invention can take the form of an entirely hardware embodiment, an entirely software (including firmware, resident software, micro-code, etc.) embodiment, or an embodiment containing both software and hardware aspects. Furthermore, the present invention can take the form of a computer program product on a computer-usable or computer-readable storage medium having computer-usable program code means embodied in the medium for use by or in connection with an instruction execution system. In the context of this document, a computer-usable or computer-readable medium can be any means that can contain, store, communicate, propagate, or transport the program for use by

or in connection with the instruction execution system, apparatus, or device.

The computer-usable or computer-readable medium can be, for example but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, device, or propagation medium. More specific examples (a nonexhaustive list) of the computer-readable medium would include the following: an electrical connection having one or more wires, a portable computer diskette, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an optical fiber, and a portable compact disc read-only memory (CDROM). Note that the computer-usable or computer-readable medium could even be paper or another suitable medium upon which the program is printed, as the program can be electronically captured, via, for instance, optical scanning of the paper or other medium, then compiled, interpreted or otherwise processed in a suitable manner if necessary, and then stored in a computer memory.

Computer program code for carrying out operations of the present invention is typically written in a high level programming language such as C or C++. Nevertheless, some modules or routines may be written in assembly or machine language to optimize speed, memory usage, or layout of the software or firmware in memory. Assembly language is typically used to implement time-critical code segments.

The present invention will now be further described with reference to the block diagram illustration of an embodiment of a system for monitoring performance of a modem of FIG. **3**. As shown in the embodiment of FIG. **3**, the host system **300** is connected to a modem **310**, such as a V.90 protocol modem. While the modem **310** is illustrated in FIG. **3** as being separate from the host system **300**, it is to be understood that, in the preferred embodiment of the present invention, the modem **310** is an internal modem device contained within the host system **300**. An example of such a host system including a modem which may be modified for use according to the teachings of the present invention is the ACP modem available with the Think Pad™ Laptop computer available from International Business Machines Corporation (IBM).

The host system **300** is coupled to the modem **310** through a primary path **315** which supports communication services utilizing the modem **310**. More particularly, communications from applications executed on the host system **300** are conveyed on the primary path **315** to the modem **310** for transmission through the port **320** which, in the illustrated embodiment, provides a connection to the Public Switched Telephone Network (PSTN). Similarly, communications from a remote device by a server modem (not shown) are received from the PSTN through port **320** and provided to a destination application executing on the host system **300** by the modem **310**. The primary path **315** may be a serial link such as an RS-232 connection. It is further to be understood that the port **320** may connect to other networks including wireless networks or a broadband network. It is to be understood that when connected with a wireless network the modem **310** may be a wireless modem. Similarly, when connected with a broadband network, the modem **310** may be a cable modem, an Asymmetric Digital Subscriber Line (ADSL), a Symmetric Digital Subscriber Line (SDSL), a High Speed Digital Subscriber Line (HDSL) or a Very High Speed Digital Subscriber Line (VDSL).

As further shown in the embodiment of FIG. **3**, the modem **310** includes a digital signal processor (DSP) **340** and an associated DSP memory **345**. The DSP **340** is coupled to the DSP memory **345** over the DSP system bus **350**. As used herein, references to the DSP memory **345** associated with the DSP **340** refer to the memory or memories within the modem **310** which are utilized for data storage by the DSP **340** during communication operations of the modem **310** supporting an active connection. This memory may include a separate memory device coupled to the DSP **340** over the DSP bus **350** and may further include memory which is contained within the circuit device of DSP **340** which is nonetheless available over the DSP system bus **350**. There also may be a multitude of DSPs being monitored if more than over DSP is used to implement the modem as, for example, with some broadband and wireless modems.

The DSP memory **345** further includes one or more first-in first-out (FIFO) buffers **355**, **360**. The FIFO buffers **355**, **360** implemented in the DSP memory **345** are used to record state transitions made for one or more of the state machines of the modem **310** as will be described further later herein. As used herein, the term "FIFO buffer" includes both circular buffers and other types of FIFO buffers.

The host system **300** is further connected to the modem **310** through the bus interface **325** as shown in the embodiment of FIG. **3**. A secondary path **335** is thereby provided for accessing the DSP memory **345** through the DSP system bus **350** by the bus interface **325** as will be described further later herein. The primary path **315** is used to support a communication connection by the modem **310** while the secondary path **335** through the bus interface **325** allows the host system **300** to access the DSP memory **345** to obtain data related to performance of the modem **310** during an active communication session supported by the primary path **315** to the modem **310**. More particularly, the bus interface **325** provides a connection from the host system bus **330** of the host system **300** to the DSP system bus **350** of the modem **310**. A secondary path **335** can also be provided through other means, for example, to provide for implementation of the systems and methods of the present invention where external modems are used to support the host system **300**. For example, as will be understood by those of skill in the art, the host system bus **330** may be coupled to an external communication port, such as a serial port or a parallel port, which provides a separate cabled link to an external modem allowing a secondary path of communication and access to the external modem while the primary coupling to the external modem is used to support a communication connection. However, it is preferred that an embodiment such as that illustrated in FIG. **3** be utilized as a direct connection between the host bus **330** and the DSP system bus **350** utilizing the bus interface **325** is expected to provide superior performance characteristics to those which would be expected using an external communicational link. Accordingly, in preferred embodiments of the present invention, modem performance is monitored by a host system **300** containing an internal modem **310**. Nonetheless, the benefits of the present invention may also be obtained in various other embodiments including those in which the secondary path **335** does not return to the same host as the primary path **315**. A second host may be co-located or remote from the first host. In fact, a remote second host could be at a distant location monitoring a modem connection through the secondary path **335**.

The DSP **340** is coupled to the FIFO buffers **355**, **360** and is configured to place internal state information associated with state machines of the modem **310** in the FIFO buffers **355**, **360**. Each of the FIFO buffers **355**, **360** may be associated with a selected one of a plurality of state

machines of the modem **310** and the DSP **350** is configured to place internal state information associated with the plurality of state machines in a corresponding one of the FIFO buffers **355**, **360**. Alternatively, a single FIFO buffer may be provided for all state machines of the modem **310** and a state tag may be provided to identify the associated state machine for each transition record in the FIFO buffer. Accordingly, the host system **300** coupled through the bus interface **325** to the DSP system bus **350** and accessing the DSP memory **345** including the FIFO buffers **355**, **360** provides a means for obtaining data related to the performance of the modem **310** from the DSP memory **345** during an active connection to the modem **310** asynchronously with communication operations of the modem **310** supporting the active connection. As used herein, the term "asynchronously" in this context means that memory access operations to read data from the DSP memory **345** and the FIFO buffers **355**, **360** to monitor performance occur independently from operations of the active connection. For example, the memory reads in a preferred embodiment are performed by a data acquisition module executing on the host system **300** independent of the code executed by the DSP **340** that supports the active communication connection through the modem **310**. More particularly, an application, such as a Windows™ application, may be executed on a host system **300**, such as an IBM Think Pad mobile computer, to record performance data obtained according to the present invention. Accordingly, improved diagnostic capabilities may be provided through the ability to access the DSP memory **345** while the modem **310** is in operation and without interfering with or interrupting data flow over the primary path **315** to the modem **310**.

Performance information so obtained may include a variety of information including the time of day, phone number dialed, call setup return codes (CSR CODE) such as those available on Microsoft Corporation's AT code #UD (UniModem diagnostic command specification), multi-phase startup procedure information such as phase **1** negotiation parameters (pursuant, for example, in a V.90 modem, to the V.8 protocol), phase **2** through **4** errors, transmit and receive speeds, retrains, disconnect errors, phone line characteristics, connecting modem brand information, power levels, and other protocol specific information. For example, with reference to a V.90 protocol, protocol specific information may include digital discontinuity impairments, equalizer coefficients, echo canceller coefficients, PADs and RBS (Rob Bit Signaling) intervals, non-linear distortion measurements and other information which will be known to those of skill in the art and as further described in the standards for modem communication, such as the V.90 standard from the ITU. This type of information may be obtained and stored by the host system **300** and subsequently used for activities, such as characterizing the telephone network to which the port **320** is coupled, including individual line conditions and other characteristics, which could be a source of complaints from modem users. The acquired data may further be utilized to analyze the capability of a particular user connection and/or to determine if a connection exhibits digital discontinuity which, for example, would typically force a V.90 modem to fall back to V.34 standard operations. Accordingly, it is to be understood that, while operations will be described herein primarily with reference to a V.90 standard, the present invention may be applied beneficially to a variety of different modem types, including both voice band and broadband communication modems, as will be known to those of skill in the art. It is further to be understood, however, that the teachings of the present invention are particularly directed to environments in which both a primary path and a secondary path are available to the DSP memory **345** to provide for monitoring operations to occur in real time while a communication connection is active through the modem.

As is evident from the types of information identified above which may be monitored according to the present invention, a significant amount of performance information can be tracked during a communication connection, for example, on a minute-by-minute basis or responsive to detection of the occurrence of certain events. The monitoring system of the present invention may be utilized to monitor internal states of the modem **310** or state transitions of one or more state machines implemented within the modem **310** and to selectively record specified parameters out of the total set of parameters available within the DSP memory **345** during state conditions where the selected parameters are significant or of potential interest to a diagnostic user. Accordingly, the types of information obtained may not only be triggered by a state transition but may further vary depending upon the state into which the respective state machine of the modem **310** has transitioned. Information may be collected on a real time basis and recorded during the life of a connection. Furthermore, information about disconnects may be gathered and throughput for a connection can be estimated.

In addition, data may also be collected when a connection is being attempted, in other words, during the startup phases before a connection is in use for data communication. Such data during startup may be especially useful when diagnosing "failure to connect" problems. Furthermore, as performance information may be collected on a real-time basis during a connection, pertinent data may be preserved which might otherwise be lost as a result of an event causing diagnostic data in the DSP memory **345** to be overwritten (for example, during retrains). The performance data may be recorded while the user of the client modem **310** is actively connected to a remote server modem in a normal manner such as through a service provider end user application (e.g. AOL, IGN Dialer and Windows Dial-up Networking) executing on the host system **300**. Performance data may be obtained throughout the active connection operations including both the startup phases and during data communication as well as the disconnect procedures. Accordingly, as used herein, the term "active connection" is intended to encompass not only the data communication portion of a connection but also the startup phases of operation implemented through the modem **310** and the corresponding disconnect phases.

A further advantage provided by the systems, methods and computer program products of the present invention is that the data collection techniques may be utilized to obtain performance data which is specific for each user connection. The state variables and other performance information collected may be unique for a particular situation, in other words, state variables may be collected for each specific client modem to server modem pair for a given point to point connection. Accordingly, when user complaints are received, state variables and other performance data may be provided for analysis of the conditions leading to the user complaint.

Utilization of the FIFO buffers **355**, **360**, such as in the illustrated embodiment of FIG. **3**, provides further advantages in monitoring performance data. The use of FIFO buffers **355**, **360** may allow the performance monitoring application executing on the host system **300** to capture every state transition for the various state machines within

the modem **310** even when access latency in reading the DSP memory **345** is longer than the duration of some states. In addition, time critical transient data may also be placed in the FIFO buffers **355**, **360** and further may be only conditionally stored based on specific state transitions within the modem **310**. Furthermore, other performance parameter data which is not as transient may also be conditionally traced on specific state transitions within the modem **310** by reading the parameters directly from the DSP memory **345** without the use of the FIFO buffers **355**, **360**. Finally, additional FIFO buffers could be implemented within the DSP memory **345** for use in maintaining highly transient performance data in addition to the FIFO buffers **355**, **360** which are associated with various state machines of the modem **310** and are used for tracking state transitions.

A particular example of a type of performance data which may be obtained by utilization of the present invention is impairment data such as that specified in the TIA PN 3857 draft 10 specification entitled "North American Telephone Network Transmission Model for Evaluating Analog Client to Digitally Connected Server Modems." Various impairments which may be wholly or fully monitored according to the teachings of the present invention in the context of a V.90 modem are found in Table 2-A of the TIA TN 3857 draft 10 document. Further, in Table 1 below, the various types of impairments from the draft 10 document and an embodiment of monitoring of the various impairments according to the present invention are provided:

TABLE 1

Type of Impairment	Illustrative Monitoring
Digital PAD Loss from Server to Client	dB measurement accurate to 3 decimal digits of precision and whether it is single or tandemn
Robbed Bit Signalling (RBS) from Server to Client and before the Digital PAD	all 6 intervals are identified so precise counts can be determined (DIL data also available)
Robbed Bit Signalling (RBS) from Server to Client and after the Digital PAD	all 6 intervals are identified so precise counts can be determined (DIL data also available)
Transhybrid Loss	precise echo canceller response is available to examine frequency dependent echo response
IMD 2nd Order	line probing reveals intermod. Estimates of distortion at 900, 1200, 1800, and 2400 Hz
Round Trip Delay	accurate to 1 millisecond
Analog PAD Loss	examination of the feedforward section of the equalizer, in conjunction with analysis of the digital PAD and analog AGC value, reveals the combined effect of loop length and analog PAD
Loop Noise	in the absence of significant nonlinear distortion, the MSE provides a good estimate of the noise

The various impairments described in Table 1 above may further be combined with information related to various cable loop lengths for the connection. A combined channel response may then be estimated for the remote central office transfer function (including digital to analog post filtering) and, the transfer function of the local loop from the central office. The transfer function for the front end of the client modem **310** may further be determined. These various transfer functions may be estimated by capturing the decision feed back equalizer's coefficients from within the client modem **310** after equalizer training is completed. In addition, an approximation of these transfer function may be examined prior to equalizer training by examining a curve fitting error which fits various V.34 modulation and carrier combinations into the measured frequency response of the line probing sequence contained in phase 2 of V.90 modem startup protocol. This information may be useful, for example, in cases where a V.90 modem falls back to V.34 operation due to digital discontinuity or an excessively long local loop.

Operations according to an embodiment of the present invention will now be described with reference to the flow chart illustration of FIG. **4**. Operations begin at block **400** with the initiation of a primary path connection by an application executing on the host system **300** to establish a communication connection through the modem **310** to a remote server modem. A secondary connection is also set up for monitoring performance of the modem **310** through the bus interface **325** (block **405**). In addition, acquisition of state information is begun utilizing the FIFO buffers **355**, **360** to track the state of one or more state machines implemented within the modem **310** (block **410**). The modem **310** typically includes a plurality of state machines each state machine having a plurality of associated states. The internal state information regarding the respective states is maintained by placement in the FIFO buffers **355**, **360**. In one embodiment, a respective one of FIFO buffers is associated with each of the different state machines of the modem **310**.

As illustrated embodiment of FIG. 4, data collection is, in part, based on timed data events. Detection of a timed data event (block **420**) results in the obtaining of performance data (block **425**). For example, specified performance parameters, which may be obtained from the DSP memory **345**, may be repeatedly obtained and stored on a periodic basis such as on a minute-by-minute basis during an active connection. In addition, a state transition event may be detected (block **430**) which also results in the acquisition and storage of selected performance data. More particularly, when a state transition event is detected (block **430**) the data associated with the occurring event is identified (block **435**). The identified categories of performance data are then obtained (block **440**). As illustrated at block **445**, the obtained performance data from either block **425** or block **440** is then stored, preferably in a text file format (block **445**).

As described previously with reference to FIG. **3**, the obtained data related to the performance of the modem is obtained from the DSP memory **345** during an active connection of the modem **310** using a secondary path **335** to the DSP memory **345**. The obtained data may correspond to startup phase operations or data communication portions of the active connection. The data related to the performance of the modem may be selected from line condition data, such as current mean squared error, current number of error events since the connection began, current data rates for transmit and receive and interworking data, such as remote retrain or rate renegotiation requests, remote modem brand identification, remote modem requests to go to error recovery during startup, retrain or rate renegotiation procedures, etc. Examples of such data in the context of a V.90 modem have been described more fully above with reference to the discussion of FIG. **3**.

The state transition event is detected at block **430** based on internal state information obtained from the FIFO buffers **355**, **360**. For example, a state transition may be detected for one of the plurality of state machines of the modem **310** based on detecting a change from one of the plurality of associated states of the one of the plurality of state machines to another of the plurality of associated states of the one of the plurality of state machines. Furthermore, operations for determining the selected data to obtain at block **435** responsive to a state transition event may be based on the one of plurality of state machines which has undergone a state transition and may further be based upon the starting and ending states of the respective state transition for that one of the plurality of state machines.

In a preferred embodiment of the present invention, the performance data is stored at block **445** in two separate text output files. The first file is utilized for a global summary of the active connection with time interval (such as minute-by-minute) snap shots detailing the value of the various performance data obtained from the DSP memory **345**. A second output file is preferably state machine focused and details the condition of the various state machines of the modem **310** and further includes any critical state variables and associated other performance data which is selected for capture based on particular state transitions.

The performance data files from block **445** may further be provided to a remote location for analysis (block **450**). Accordingly, a manufacturer or service provider may remotely access diagnostic information suitable for analyzing problems and addressing user complaints. More particularly, connection specific information may be provided which may be particularly useful in light of the differences between line connections for various users in point to point connections thereby allowing the service provider to deal with individual cases to provide responsive support to individual users. It is further to be understood that the operations at blocks **420** through **450** may repeat throughout the duration of a connection as additional timed data events or state transition events are encountered or until monitoring operations are discontinued by a user.

In order to aid those of skill in the art to further understand the present invention, exemplary performance monitoring outputs in a two file format for an embodiment of the present invention as shown in Appendix A and Appendix B will now be described. Appendix A and Appendix B correspond respectively to the two output text file format described for a preferred embodiment above. More particularly, Appendix A is an output file which is state machine driven and details the state machines of a modem and identifies critical state variables that are captured based on state transitions. As shown in Appendix A, the respective state machines of the modem include four separate state machines: transmit (TX), receive (RX), front end transmit (FT) and front end receive (FR). The various states for each of the respective state machines correspond to operations in compliance with the V.90 standard and will be understood by those of skill in the art familiar with that standard. The first column indicates the time of occurrence of the state condition. The asterisks in the state machine columns indicate the state machine which underwent a transition triggering the creation of the row entry. The final column does not correspond to a state machine but reflects a data value for the current mean squared error (MSE) at the output of the equalizer which the exemplary protocol captures and records at each state transition. In addition, at various points in the state machine tracking text file of Appendix A, selected performance data values are captured responsive to particular state machine state transitions. For example, at the time 13: 23: 44, responsive to a transition of the RX state machine from a recv_J state to a recv_JP state, various equalizer and echo canceller related parameter values are captured. State transition triggered parameter capture is also reflected at times 13: 23: 47, 13:23: 49, 13:23:50, 13:23: 52 and 13: 23: 55.

The second text output file format is illustrated by the exemplary embodiment in Appendix B. Appendix B corresponds to a text file which is a global summary of a connection with details. As with the example of Appendix A, the terminology utilized in the example of Appendix B is consistent with that defined by the V.90 standard and will be understood by those of ordinary skill in the art. Prior to the power levels section is a list of a number of V.8 (phase 1) bytes from both the local and the remote modem. The power levels are self explanatory. The global minute by minute summary includes LMCPS (last minute characters per second) which shows the estimated average throughput of the modem during the last minute including the effects of retrains, rate renegotiations and line errors. SUMEE refers to the running sum of line errors that have taken place since the connection began. The remaining parameters are generally related to a particular ACP modem, the ThinkPad™ ACP modem, and may or may not be parameters found in other modem products.

The present invention has been described above with reference to the block diagram illustrations of FIG. **3**. and the flowchart illustration of FIG. **4**. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These program instructions may be provided to a processor to produce a machine, such that the instructions which execute on the processor create means for implementing the functions specified in the flowchart or block diagram block or blocks. The computer program instructions may be executed by a processor to cause a series of operational steps to be performed by the processor to produce a computer implemented process such that the instructions which execute on the processor provide steps for implementing the functions specified in the flowchart or block diagram block or blocks.

Accordingly, blocks of the block diagrams and/or flowchart illustrations support combinations of means for performing the specified functions, combinations of steps for performing the specified functions and program instruction means for performing the specified functions. It will also be understood that each block of the block diagrams and/or flowchart illustrations, and combinations of blocks in the block diagrams and/or flowchart illustrations, can be imple-

mented by special purpose hardware-based systems which perform the specified functions or steps, or combinations of special purpose hardware and computer instructions.

It should also be noted that, in some alternative implementations, the functions noted in the blocks may occur out of the order noted in the figures. For example, two blocks shown in succession may in fact be executed substantially concurrently or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved.

While the present invention has been illustrated and described in detail in the drawings and foregoing description, it is understood that the embodiments shown are merely exemplary. Moreover, it is to be understood that many variations and modifications can be made to the embodiments described herein above without substantially departing from the principles of the present invention. All such variations and modifications are intended to be included herein within the scope of the present invention, as set forth in the following claims.

We claim:

1. A method for monitoring performance of a modem comprising the steps of:

establishing an active communications connection utilizing the modem;

obtaining data related to the performance of the modem, including internal state information, from a digital signal processor (DSP) memory of the modem using a secondary path to the DSP memory which is separate and independent from a primary path to said DSP memory utilized in support of the communications connection;

determining that a state transition has occurred based on the internal state information; and capturing a selected type of data related to the performance of the modem responsive to a state transition.

2. A method according to claim **1** wherein the obtaining step further comprises the step of obtaining data related to the performance of the modem from the DSP memory during startup of the active connection.

3. A method according to claim **1** wherein the data related to the performance of the modem is selected from the group consisting of line condition data and interworking data.

4. A method according to claim **1** wherein the modem includes a plurality of state machines each state machine having a plurality of associated states and wherein the determining step further comprises the step of determining that a state transition of one of the plurality of state machines has occurred based on detecting a change from one of the plurality of associated states of the one of the plurality of state machines to another of the plurality of associated states of the one of the plurality of state machines and wherein the selected type of data is selected based on the one of the plurality of state machines.

5. A method according to claim **4** wherein the selected type of data is further selected based on the one of the plurality of associated states and the another of the plurality of associated states.

6. A method according to claim 1 wherein the determining step is preceded by the step of providing a first-in first-out (FIFO) buffer in the DSP memory and wherein the determining step further comprises the step of placing the internal state information in the FIFO buffer.

7. A method according to claim **6** wherein the providing step further comprises the step of providing a plurality of FIFO buffers in the DSP memory, each of the plurality of FIFO buffers being associated with a different state machine of the modem and wherein the placing step further comprises placing internal state information associated with a respective state machine of the modem in a corresponding one of the plurality of FIFO buffers.

8. A method according to claim **1** wherein the obtaining step is followed by the step of storing the data related to the performance of the modem in a text file.

9. A method according to claim **8** wherein the storing step is followed by the step of providing the stored data to a remote location for analysis.

10. A system for monitoring performance of a modem comprising:

a host system having a host system bus;

a modem, the modem including:

a digital signal processor (DSP);

a memory coupled to the DSP over a DSP system bus;

a primary path to the DSP memory that supports a communication connection;

a plurality of first-in first-out (FIFO) buffers coupled to the DSP, each of the FIFO buffers being associated with one of a plurality of state machines of the modem and

wherein the DSP is configured to place internal state information associated with the

plurality of state machines in a corresponding one of the FIFO buffers; and

a bus interface coupling the host system bus to the DSP system bus, the bus interface being configured to allow the host system to access the DSP memory to obtain data related to performance of the modem during an active communication session supported by the primary path of the modem using a secondary path to the DSP memory which is separate and independent from the primary path to said DSP.

11. A system for monitoring performance of a modem comprising:

an interface coupled to the modem;

means for obtaining data related to the performance of the modem, including internal state information, from a digital signal processor (DSP) memory of the modem during an active connection to the modem using a secondary path to the DSP memory which is separate and independent from a primary path to said DSP memory utilized in support of the communications connection, the secondary path including the interface coupled to the modem;

means for determining that a state transition has occurred based on the internal state information; and

means for capturing a selected type of data related to the performance of the modem responsive to a state transition.

12. A system according to claim **11** wherein the means for obtaining further comprises means for obtaining data related to the performance of the modem from the DSP memory during startup of the active connection.

13. A system according to claim **11** wherein the data related to the performance of the modem is selected from the group consisting of line condition data and interworking data.

14. A system according to claim **11** wherein the modem includes a plurality of state machines each state machine having a plurality of associated states and wherein the means for determining further comprises means for determining that a state transition of one of the plurality of state machines has occurred based on detecting a change from one of the

plurality of associated states of the one of the plurality of state machines to another of the plurality of associated states of the one of the plurality of state machines and wherein the selected type of data is selected based on the one of the plurality of state machines.

15. A system according to claim **14** wherein the selected type of data is further selected based on the one of the plurality of associated states and the another of the plurality of associated states.

16. A system according to claim **11** wherein the system further comprises a first-in first-out (FIFO) buffer in the DSP memory and wherein the means for determining further comprises means for placing the internal state information in the FIFO buffer.

17. A system according to claim **16** wherein the means for providing further comprises means for providing a plurality of FIFO buffers in the DSP memory, each of the plurality of FIFO buffers being associated with a different state machine of the modem and wherein the means for placing further comprises means for placing internal state information associated with a respective state machine of the modem in a corresponding one of the plurality of FIFO buffers.

18. A system according to claim **11** further comprising means for storing the date related to the performance of the modem in a text file.

19. A system according to claim **18** further comprising means for providing the stored data to a remote location for analysis.

20. A computer program product for monitoring performance of a modem, comprising:

a computer readable storage medium having computer readable program code means embodied therein, the computer readable code means comprising:

computer readable code which establishes an active communications connection utilizing the modem;

computer readable code which obtains data related to the performance of the modem, including internal state information, from a digital signal processor (DSP) memory of the modem using a secondary path to the DSP memory which is separate and independent from a primary path to said DSP memory utilized in support of the communications connection;

computer readable code which determines that a state transition has occurred based on the internal state information; and

computer readable code which captures a selected type of data related to the performance of the modem responsive to a state transition.

21. A computer program product according to claim **20** wherein the computer readable code which obtains further comprises computer readable code which obtains data related to the performance of the modem from the DSP memory during startup of the active connection.

22. A computer program product according to claim **20** wherein the data related to the performance of the modem is selected from the group consisting of line condition data and interworking data.

23. A computer program product according to claim **20** wherein the modem includes a plurality of state machines each state machine having a plurality of associated states and wherein the computer readable code which determines further comprises computer readable code which determines that a state transition of one of the plurality of state machines has occurred based on detecting a change from one of the plurality of associated states of the one of the plurality of state machines to another of the plurality of associated states of the one of the plurality of state machines and wherein the selected type of data is selected based on the one of the plurality of state machines.

24. A computer program product according to claim **23** wherein the selected type of data is further selected based on the one of the plurality of associated states and the another of the plurality of associated states.

25. A computer program product according to claim **20** wherein the computer readable code which determines further comprises computer readable code which places the internal state information in a first-in first-out (FIFO) buffer in the DSP memory.

26. A computer program product according to claim **25** wherein the computer readable code which provides further comprises computer readable code which provides a plurality of FIFO buffers in the DSP memory, each of the plurality of FIFO buffers being associated with a different state machine of the modem and wherein the computer readable code which places further comprises computer readable code which places internal state information associated with a respective state machine of the modem in a corresponding one of the plurality of FIFO buffers.

27. A computer program product according to claim **20** further comprising computer readable code which stores the data related to the performance of the modem in a text file.

28. A computer program product according to claim **27** further comprising computer readable code which provides the stored data to a remote location for analysis.

* * * * *

EXHIBIT 2

US006467052B1

(12) **United States Patent**
Kaler et al.

(10) **Patent No.: US 6,467,052 B1**
(45) **Date of Patent: Oct. 15, 2002**

(54) **METHOD AND APPARATUS FOR ANALYZING PERFORMANCE OF DATA PROCESSING SYSTEM**

(75) Inventors: **Christopher G. Kaler**, Redmond; **Martyn S. Lovell**; **Robert S. Wahbe**, both of Seattle; **William J. Ferguson**, Bellevue, all of WA (US); **Oliver J. Sharp**, New York, NY (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/325,469**

(22) Filed: **Jun. 3, 1999**

(51) **Int. Cl.**[7] **G06F 11/34**

(52) **U.S. Cl.** 714/39; 714/47; 709/318; 717/127; 717/130

(58) **Field of Search** 714/39, 47, 57, 714/46; 709/318; 717/124, 127, 128, 130

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,745,693 A * 4/1998 Knight et al. 709/224
5,752,159 A * 5/1998 Faust et al. 455/5.1
5,768,614 A * 6/1998 Takagi et al. 710/1
6,138,121 A * 10/2000 Costa et al. 707/100
6,243,838 B1 * 6/2001 Liu et al. 714/57
6,249,755 B1 * 6/2001 Yemini et al. 702/183
6,314,533 B1 * 11/2001 Novik et al. 714/39

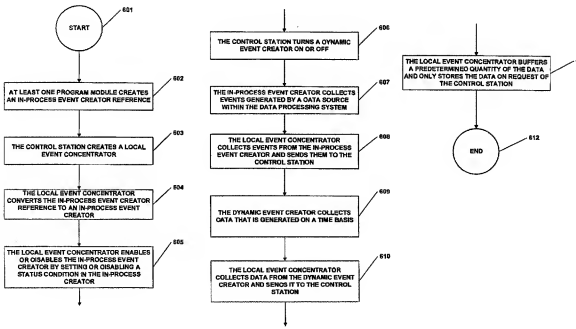
* cited by examiner

Primary Examiner—Scott Baderman
(74) *Attorney, Agent, or Firm*—Woodcock Washburn LLP

(57) **ABSTRACT**

A method and apparatus for analyzing the performance of a data processing system, particularly a distributed data processing system, provide a system user with tools for analyzing an application running thereon. Information about the flow and performance of the application can be specified, captured, and analyzed, without modifying it or degrading its performance or data security characteristics, even if it is distributed across multiple machines. The user interface permits the system user to filter the performance information, to set triggers which the performance analyzer is able to reduce and/or combine, to observe multiple time-synchronized displays of performance data either in real time or post mortem, and to play and re-play the operation of an automatically generated application model. The invention is implemented in part by providing suitable Application Program Interfaces (APIs) in the operating system of the data processing system.

31 Claims, 33 Drawing Sheets



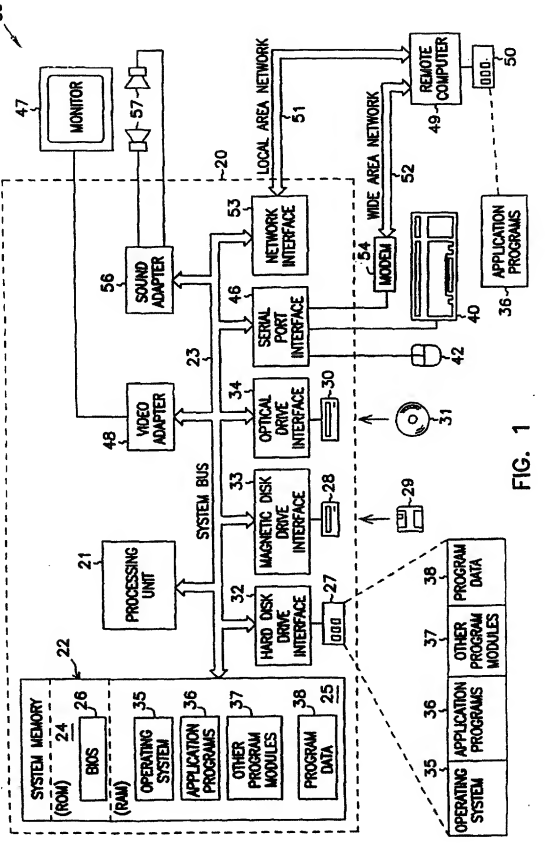

80
47
MONITOR
57
20
LOCAL AREA NETWORK
51
REMOTE COMPUTER
49
50
56
SOUND ADAPTER
53
NETWORK INTERFACE
WIDE AREA NETWORK
52
54
MODEM
APPLICATION PROGRAMS
36
46
SERIAL PORT INTERFACE
40
42
48
VIDEO ADAPTER
23
34
OPTICAL DRIVE INTERFACE
30
31
SYSTEM BUS
33
MAGNETIC DISK DRIVE INTERFACE
28
29
21
PROCESSING UNIT
32
HARD DISK DRIVE INTERFACE
27
22
SYSTEM MEMORY
(ROM) 24
BIOS 26
(RAM)
OPERATING SYSTEM 35
APPLICATION PROGRAMS 36
OTHER PROGRAM MODULES 37
PROGRAM DATA 38
25
PROGRAM DATA 38
OTHER PROGRAM MODULES 37
APPLICATION PROGRAMS 36
OPERATING SYSTEM 35

FIG. 1

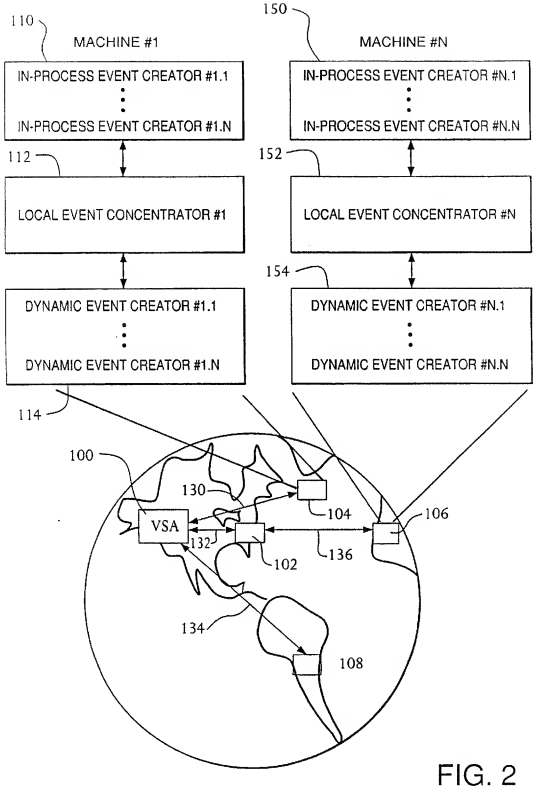
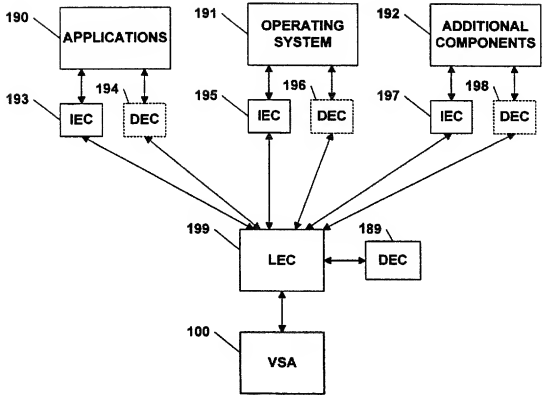

110
MACHINE #1
IN-PROCESS EVENT CREATOR #1.1
IN-PROCESS EVENT CREATOR #1.N
112
LOCAL EVENT CONCENTRATOR #1
DYNAMIC EVENT CREATOR #1.1
DYNAMIC EVENT CREATOR #1.N
150
MACHINE #N
IN-PROCESS EVENT CREATOR #N.1
IN-PROCESS EVENT CREATOR #N.N
152
LOCAL EVENT CONCENTRATOR #N
154
DYNAMIC EVENT CREATOR #N.1
DYNAMIC EVENT CREATOR #N.N
114
100
130
VSA
132
104
106
102
136
134
108

FIG. 2

FIG. 3


190
APPLICATIONS
191
OPERATING SYSTEM
192
ADDITIONAL COMPONENTS
193
IEC
194
DEC
195
IEC
196
DEC
197
IEC
198
DEC
199
LEC
189
DEC
100
VSA

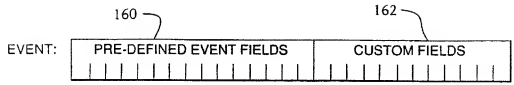


FIG. 4

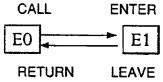


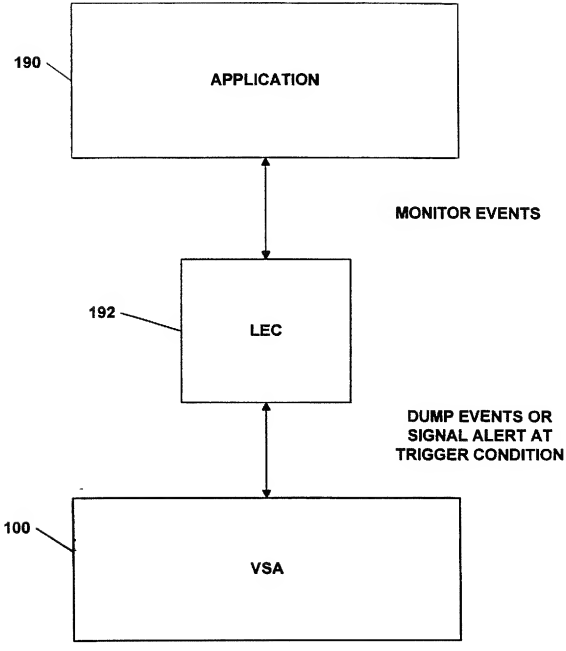
FIG. 5

		SOURCE					TARGET						
	EVENT:	MACHINE	PROCESS	ENTITY	INSTANCE	HANDLE	MACHINE	PROCESS	ENTITY	INSTANCE	HANDLE	CORRELATION	CAUSALITY
170	CALL	M0	P0	E0	I0	H0					H1	CA	UA
172	ENTER					H0	M1	P1	E1	I1	H1	CB	UA
174	LEAVE					H0	M1	P1	E1	I1	H1	CB	UA
176	RETURN	M0	P0	E0	I0	H0					H1	CB	UA

180 186 182 184

FIG.6

FIG. 7


190
APPLICATION
MONITOR EVENTS
192
LEC
DUMP EVENTS OR
SIGNAL ALERT AT
TRIGGER CONDITION
100
VSA

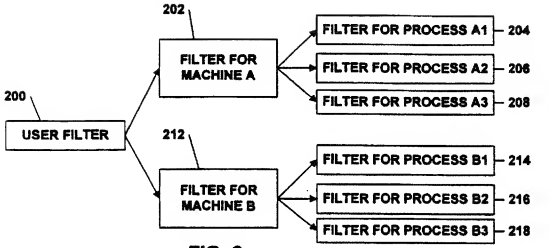


FIG. 8

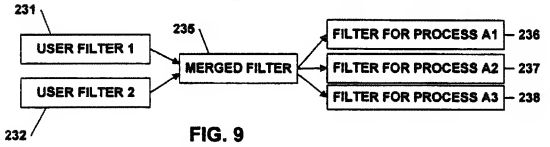


FIG. 9

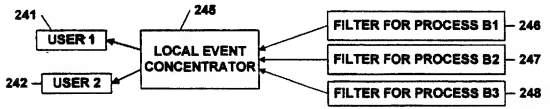


FIG. 10

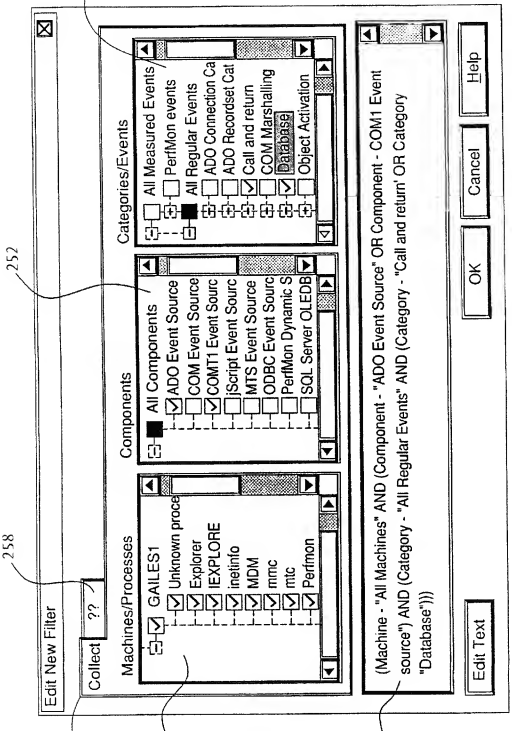

Edit New Filter
Collect
??
Machines/Processes
GAILES1
Unknown proce
Explorer
IEXPLORE
inetinfo
MDM
mmc
mtc
Perfmon
Components
All Components
ADO Event Source
COM Event Source
COMT1 Event Sourc
jScript Event Sourc
MTS Event Source
ODBC Event Sourc
PerfMon Dynamic S
SQL Server OLEDB
Categories/Events
All Measured Events
PerfMon events
All Regular Events
ADO Connection Ca
ADO Recordset Cat
Call and return
COM Marshalling
Database
Object Activation
(Machine - "All Machines" AND (Component - "ADO Event Source" OR Component - COM1 Event source") AND (Category - "All Regular Events" AND (Category - "Call and return' OR Category "Database")))
Edit Text
OK
Cancel
Help
256
258
250
252
254
260

FIG. 11

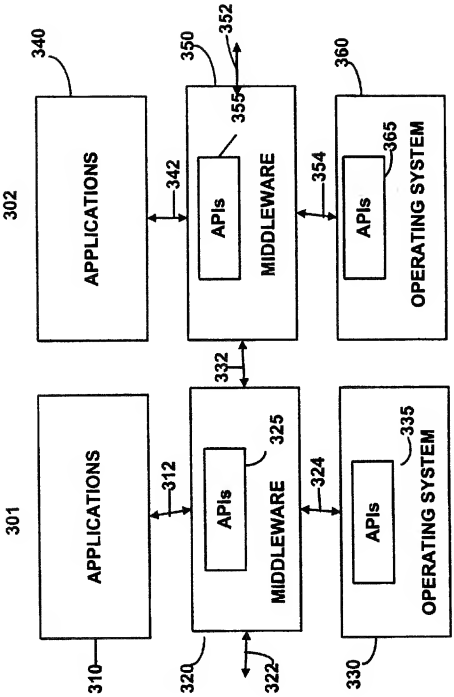

301
APPLICATIONS
310
312
APIs
MIDDLEWARE
325
320
322
324
APIs
OPERATING SYSTEM
335
330
332
302
APPLICATIONS
340
342
APIs
MIDDLEWARE
350
352
355
354
APIs
OPERATING SYSTEM
360
365

FIG. 12

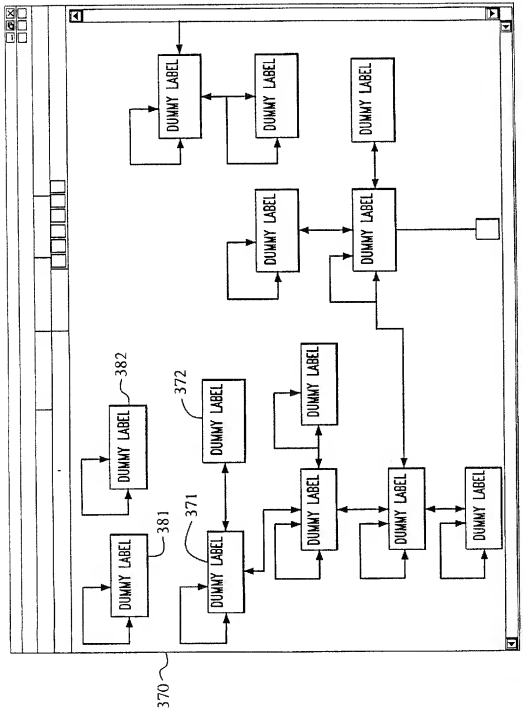


FIG. 13

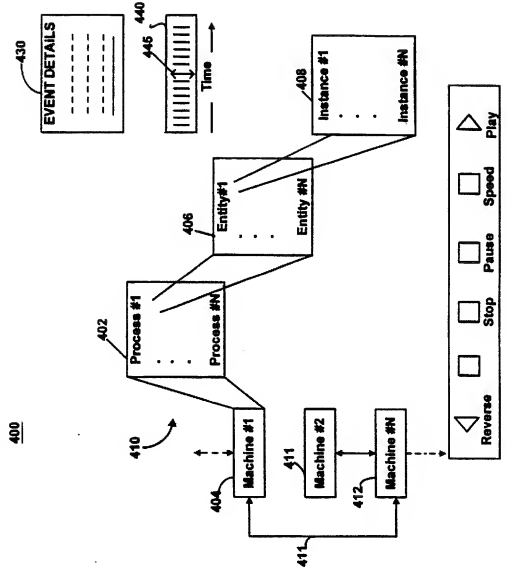

400
410
430
EVENT DETAILS
440
445
Time
402
Process #1
Process #N
406
Entity#1
Entity #N
408
Instance #1
Instance #N
Machine #1
Machine #2
Machine #N
404
411
412
Reverse
Stop
Pause
Speed
Play

FIG. 14

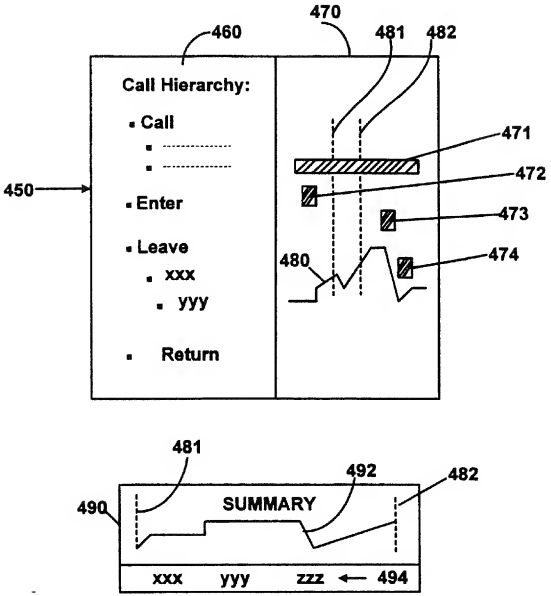

460
470
481
482
Call Hierarchy:
Call
Enter
Leave
xxx
yyy
Return
450
471
472
473
474
480
481
492
482
490
SUMMARY
xxx
yyy
zzz
494

FIG. 15

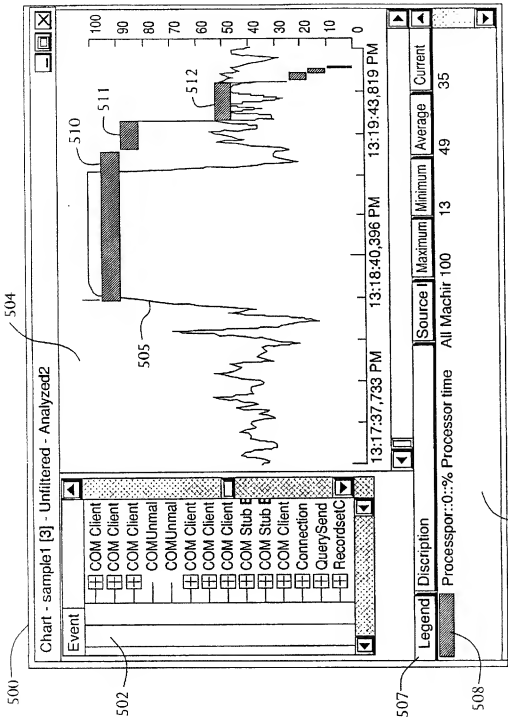

500
502
504
505
506
507
508
510
511
512
Chart - sample1 [3] - Unfiltered - Analyzed2
Event
COM Client
COM Client
COM Client
COMUnmal
COMUnmal
COM Client
COM Client
COM Client
COM Stub E
COM Stub E
COM Client
Connection
QuerySend
RecordsetC
100
90
80
70
60
50
40
30
20
10
0
13:17:37,733 PM
13:18:40,396 PM
13:19:43,819 PM
Legend
Discription
Source
Maximum
Minimum
Average
Current
Processpor::0::% Processor time
All Machir
100
13
49
35

FIG. 16

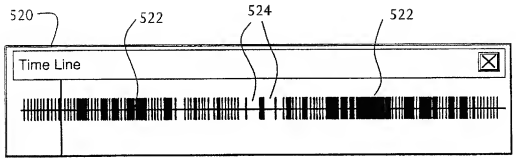

520
522
524
522
Time Line

FIG. 17

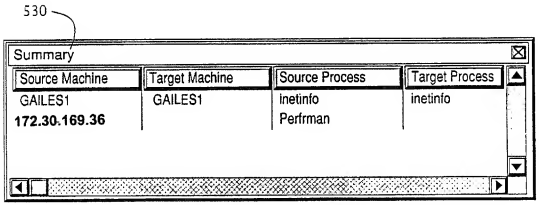

530
Summary
Source Machine
Target Machine
Source Process
Target Process
GAILES1
172.30.169.36
GAILES1
inetinfo
Perfrman
inetinfo

FIG. 18

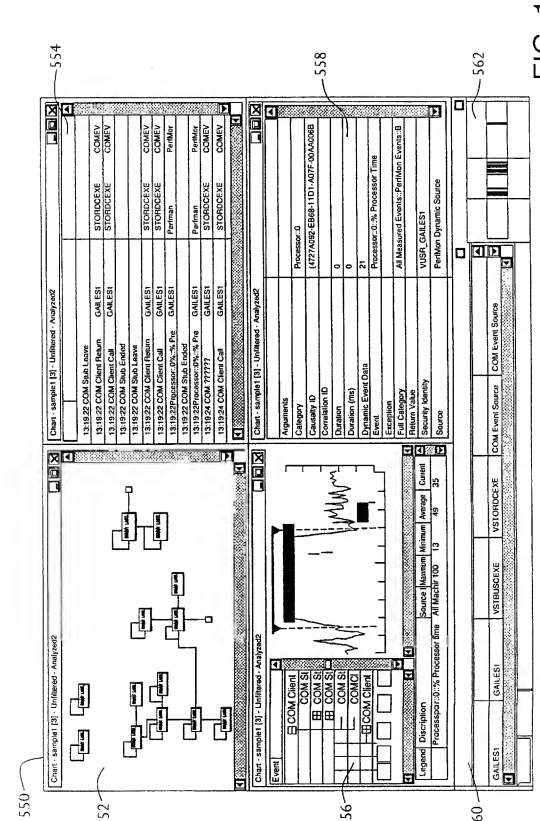


FIG. 19

FIG. 20A

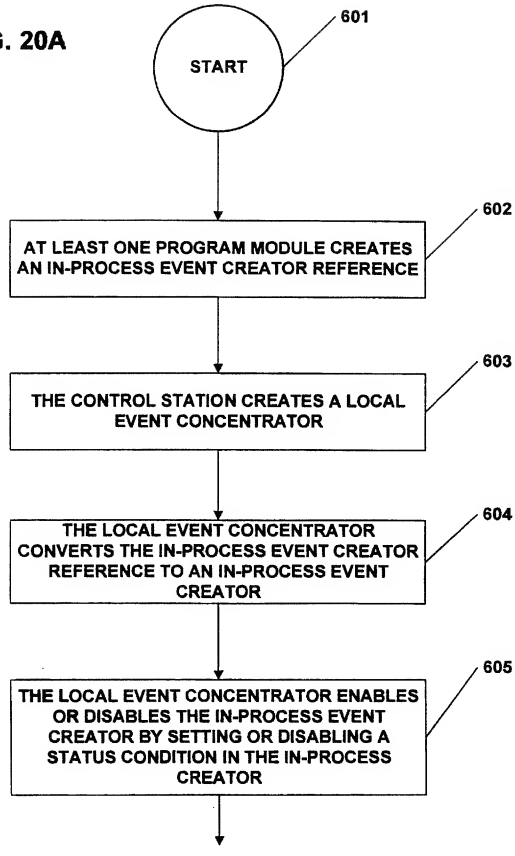

START
601
AT LEAST ONE PROGRAM MODULE CREATES AN IN-PROCESS EVENT CREATOR REFERENCE
602
THE CONTROL STATION CREATES A LOCAL EVENT CONCENTRATOR
603
THE LOCAL EVENT CONCENTRATOR CONVERTS THE IN-PROCESS EVENT CREATOR REFERENCE TO AN IN-PROCESS EVENT CREATOR
604
THE LOCAL EVENT CONCENTRATOR ENABLES OR DISABLES THE IN-PROCESS EVENT CREATOR BY SETTING OR DISABLING A STATUS CONDITION IN THE IN-PROCESS CREATOR
605

FIG. 20B

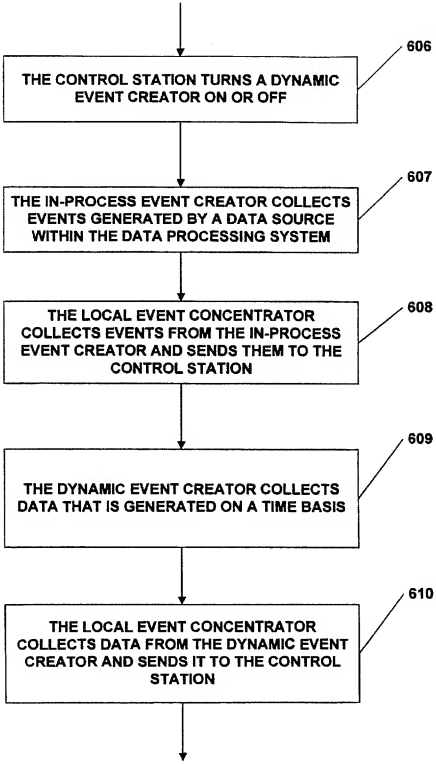

606
THE CONTROL STATION TURNS A DYNAMIC EVENT CREATOR ON OR OFF
607
THE IN-PROCESS EVENT CREATOR COLLECTS EVENTS GENERATED BY A DATA SOURCE WITHIN THE DATA PROCESSING SYSTEM
608
THE LOCAL EVENT CONCENTRATOR COLLECTS EVENTS FROM THE IN-PROCESS EVENT CREATOR AND SENDS THEM TO THE CONTROL STATION
609
THE DYNAMIC EVENT CREATOR COLLECTS DATA THAT IS GENERATED ON A TIME BASIS
610
THE LOCAL EVENT CONCENTRATOR COLLECTS DATA FROM THE DYNAMIC EVENT CREATOR AND SENDS IT TO THE CONTROL STATION

FIG. 20C

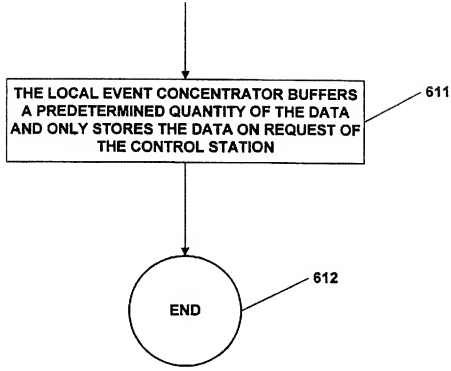

THE LOCAL EVENT CONCENTRATOR BUFFERS A PREDETERMINED QUANTITY OF THE DATA AND ONLY STORES THE DATA ON REQUEST OF THE CONTROL STATION
611
END
612

FIG. 21A

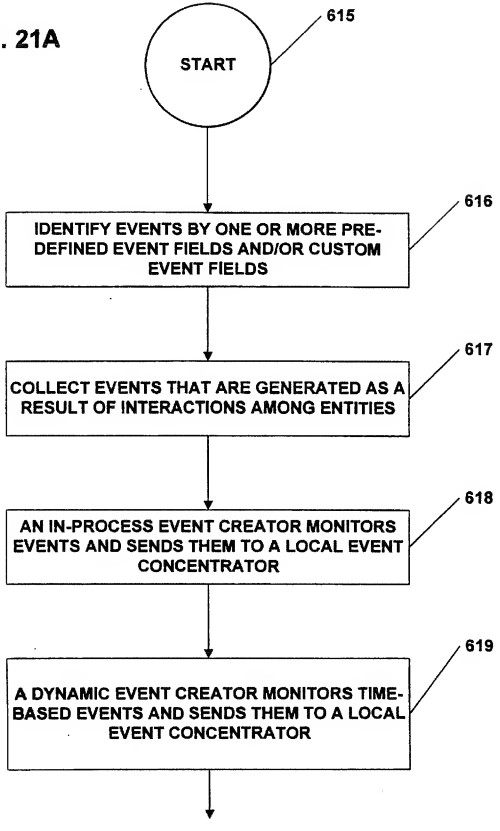

START
615
IDENTIFY EVENTS BY ONE OR MORE PRE-DEFINED EVENT FIELDS AND/OR CUSTOM EVENT FIELDS
616
COLLECT EVENTS THAT ARE GENERATED AS A RESULT OF INTERACTIONS AMONG ENTITIES
617
AN IN-PROCESS EVENT CREATOR MONITORS EVENTS AND SENDS THEM TO A LOCAL EVENT CONCENTRATOR
618
A DYNAMIC EVENT CREATOR MONITORS TIME-BASED EVENTS AND SENDS THEM TO A LOCAL EVENT CONCENTRATOR
619

FIG. 21B

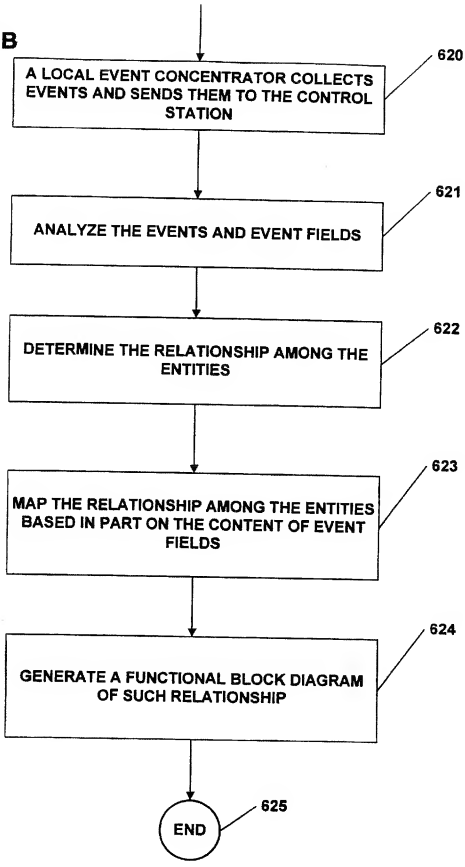

A LOCAL EVENT CONCENTRATOR COLLECTS EVENTS AND SENDS THEM TO THE CONTROL STATION
620
ANALYZE THE EVENTS AND EVENT FIELDS
621
DETERMINE THE RELATIONSHIP AMONG THE ENTITIES
622
MAP THE RELATIONSHIP AMONG THE ENTITIES BASED IN PART ON THE CONTENT OF EVENT FIELDS
623
GENERATE A FUNCTIONAL BLOCK DIAGRAM OF SUCH RELATIONSHIP
624
END
625

FIG. 22A

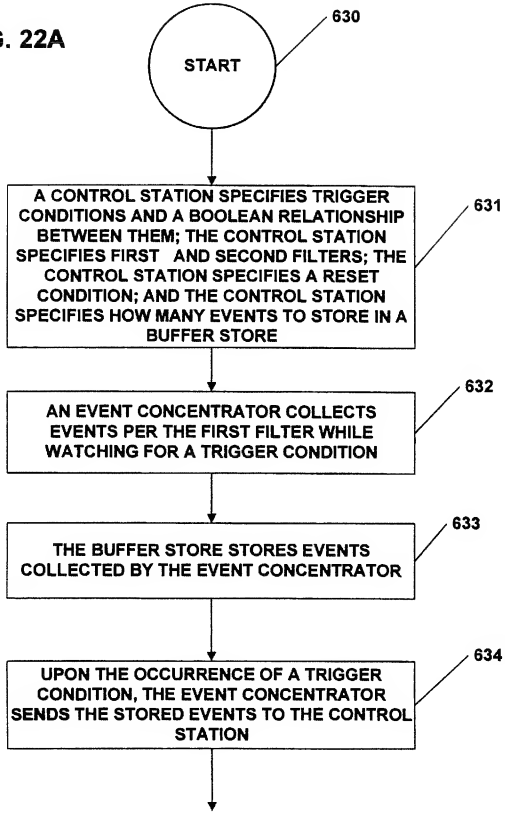

START
630
A CONTROL STATION SPECIFIES TRIGGER CONDITIONS AND A BOOLEAN RELATIONSHIP BETWEEN THEM; THE CONTROL STATION SPECIFIES FIRST AND SECOND FILTERS; THE CONTROL STATION SPECIFIES A RESET CONDITION; AND THE CONTROL STATION SPECIFIES HOW MANY EVENTS TO STORE IN A BUFFER STORE
631
AN EVENT CONCENTRATOR COLLECTS EVENTS PER THE FIRST FILTER WHILE WATCHING FOR A TRIGGER CONDITION
632
THE BUFFER STORE STORES EVENTS COLLECTED BY THE EVENT CONCENTRATOR
633
UPON THE OCCURRENCE OF A TRIGGER CONDITION, THE EVENT CONCENTRATOR SENDS THE STORED EVENTS TO THE CONTROL STATION
634

FIG. 22B

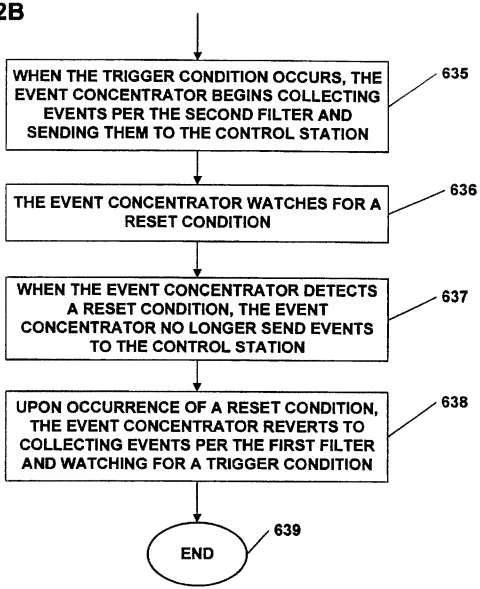

WHEN THE TRIGGER CONDITION OCCURS, THE EVENT CONCENTRATOR BEGINS COLLECTING EVENTS PER THE SECOND FILTER AND SENDING THEM TO THE CONTROL STATION
635
THE EVENT CONCENTRATOR WATCHES FOR A RESET CONDITION
636
WHEN THE EVENT CONCENTRATOR DETECTS A RESET CONDITION, THE EVENT CONCENTRATOR NO LONGER SEND EVENTS TO THE CONTROL STATION
637
UPON OCCURRENCE OF A RESET CONDITION, THE EVENT CONCENTRATOR REVERTS TO COLLECTING EVENTS PER THE FIRST FILTER AND WATCHING FOR A TRIGGER CONDITION
638
END
639

FIG. 23A

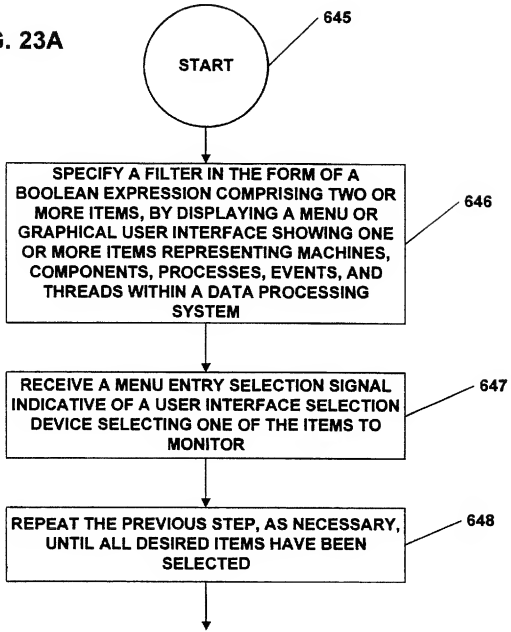

START
645
SPECIFY A FILTER IN THE FORM OF A BOOLEAN EXPRESSION COMPRISING TWO OR MORE ITEMS, BY DISPLAYING A MENU OR GRAPHICAL USER INTERFACE SHOWING ONE OR MORE ITEMS REPRESENTING MACHINES, COMPONENTS, PROCESSES, EVENTS, AND THREADS WITHIN A DATA PROCESSING SYSTEM
646
RECEIVE A MENU ENTRY SELECTION SIGNAL INDICATIVE OF A USER INTERFACE SELECTION DEVICE SELECTING ONE OF THE ITEMS TO MONITOR
647
REPEAT THE PREVIOUS STEP, AS NECESSARY, UNTIL ALL DESIRED ITEMS HAVE BEEN SELECTED
648

FIG. 23B

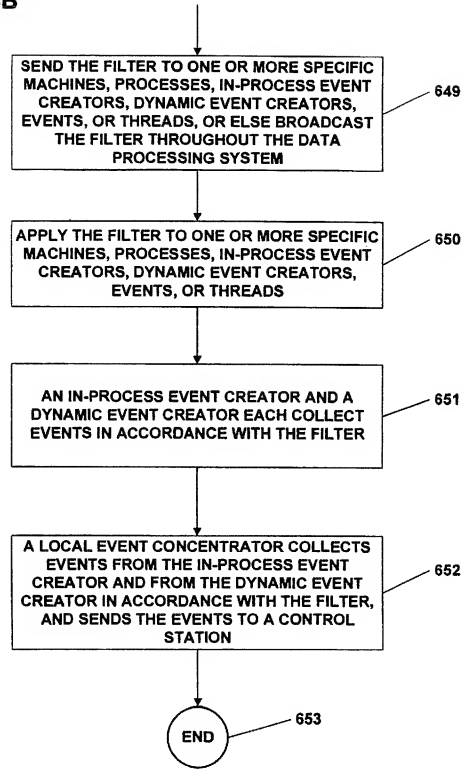

SEND THE FILTER TO ONE OR MORE SPECIFIC MACHINES, PROCESSES, IN-PROCESS EVENT CREATORS, DYNAMIC EVENT CREATORS, EVENTS, OR THREADS, OR ELSE BROADCAST THE FILTER THROUGHOUT THE DATA PROCESSING SYSTEM
649
APPLY THE FILTER TO ONE OR MORE SPECIFIC MACHINES, PROCESSES, IN-PROCESS EVENT CREATORS, DYNAMIC EVENT CREATORS, EVENTS, OR THREADS
650
AN IN-PROCESS EVENT CREATOR AND A DYNAMIC EVENT CREATOR EACH COLLECT EVENTS IN ACCORDANCE WITH THE FILTER
651
A LOCAL EVENT CONCENTRATOR COLLECTS EVENTS FROM THE IN-PROCESS EVENT CREATOR AND FROM THE DYNAMIC EVENT CREATOR IN ACCORDANCE WITH THE FILTER, AND SENDS THE EVENTS TO A CONTROL STATION
652
END
653

FIG. 24A

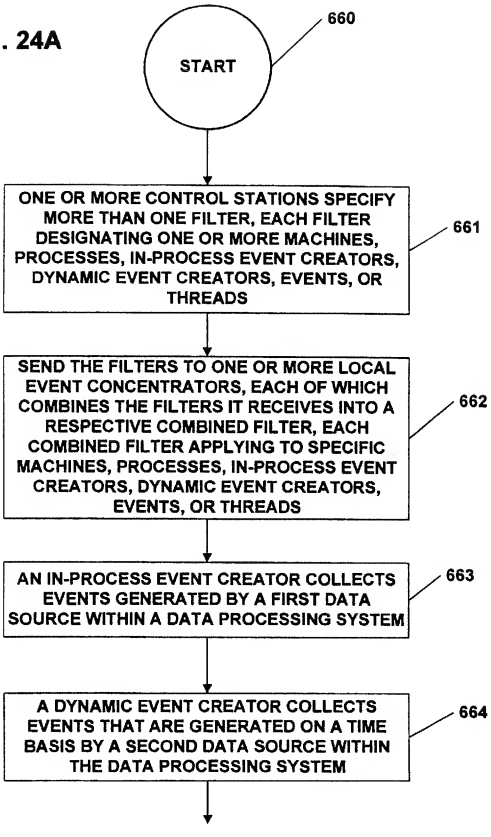

START
660
ONE OR MORE CONTROL STATIONS SPECIFY MORE THAN ONE FILTER, EACH FILTER DESIGNATING ONE OR MORE MACHINES, PROCESSES, IN-PROCESS EVENT CREATORS, DYNAMIC EVENT CREATORS, EVENTS, OR THREADS
661
SEND THE FILTERS TO ONE OR MORE LOCAL EVENT CONCENTRATORS, EACH OF WHICH COMBINES THE FILTERS IT RECEIVES INTO A RESPECTIVE COMBINED FILTER, EACH COMBINED FILTER APPLYING TO SPECIFIC MACHINES, PROCESSES, IN-PROCESS EVENT CREATORS, DYNAMIC EVENT CREATORS, EVENTS, OR THREADS
662
AN IN-PROCESS EVENT CREATOR COLLECTS EVENTS GENERATED BY A FIRST DATA SOURCE WITHIN A DATA PROCESSING SYSTEM
663
A DYNAMIC EVENT CREATOR COLLECTS EVENTS THAT ARE GENERATED ON A TIME BASIS BY A SECOND DATA SOURCE WITHIN THE DATA PROCESSING SYSTEM
664

FIG. 24B

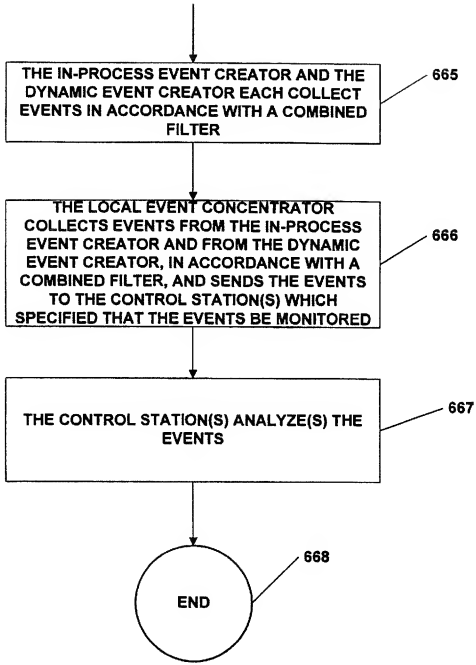

THE IN-PROCESS EVENT CREATOR AND THE DYNAMIC EVENT CREATOR EACH COLLECT EVENTS IN ACCORDANCE WITH A COMBINED FILTER
665
THE LOCAL EVENT CONCENTRATOR COLLECTS EVENTS FROM THE IN-PROCESS EVENT CREATOR AND FROM THE DYNAMIC EVENT CREATOR, IN ACCORDANCE WITH A COMBINED FILTER, AND SENDS THE EVENTS TO THE CONTROL STATION(S) WHICH SPECIFIED THAT THE EVENTS BE MONITORED
666
THE CONTROL STATION(S) ANALYZE(S) THE EVENTS
667
END
668

FIG. 25A

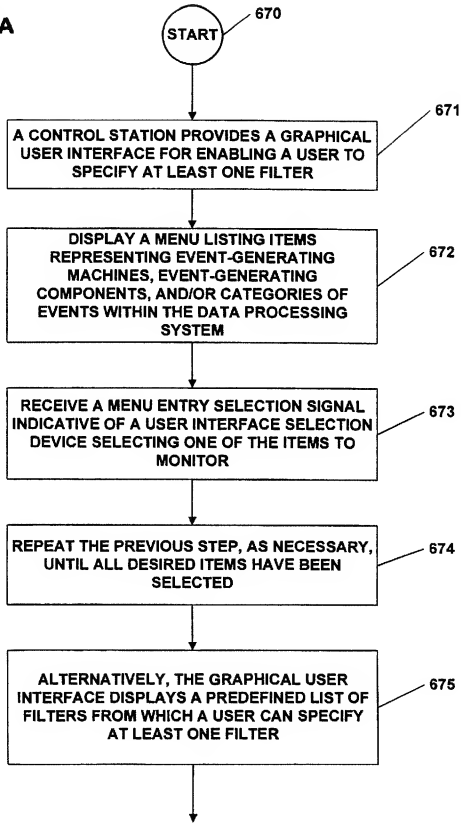

START
670
A CONTROL STATION PROVIDES A GRAPHICAL USER INTERFACE FOR ENABLING A USER TO SPECIFY AT LEAST ONE FILTER
671
DISPLAY A MENU LISTING ITEMS REPRESENTING EVENT-GENERATING MACHINES, EVENT-GENERATING COMPONENTS, AND/OR CATEGORIES OF EVENTS WITHIN THE DATA PROCESSING SYSTEM
672
RECEIVE A MENU ENTRY SELECTION SIGNAL INDICATIVE OF A USER INTERFACE SELECTION DEVICE SELECTING ONE OF THE ITEMS TO MONITOR
673
REPEAT THE PREVIOUS STEP, AS NECESSARY, UNTIL ALL DESIRED ITEMS HAVE BEEN SELECTED
674
ALTERNATIVELY, THE GRAPHICAL USER INTERFACE DISPLAYS A PREDEFINED LIST OF FILTERS FROM WHICH A USER CAN SPECIFY AT LEAST ONE FILTER
675

FIG. 25B

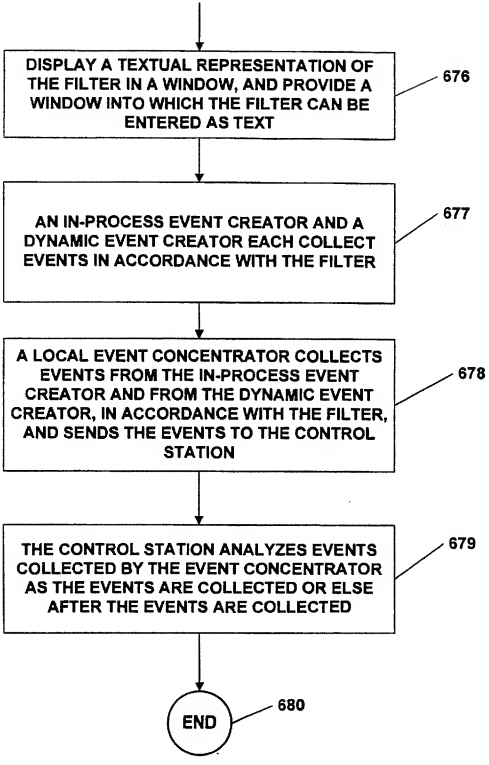

DISPLAY A TEXTUAL REPRESENTATION OF THE FILTER IN A WINDOW, AND PROVIDE A WINDOW INTO WHICH THE FILTER CAN BE ENTERED AS TEXT
676
AN IN-PROCESS EVENT CREATOR AND A DYNAMIC EVENT CREATOR EACH COLLECT EVENTS IN ACCORDANCE WITH THE FILTER
677
A LOCAL EVENT CONCENTRATOR COLLECTS EVENTS FROM THE IN-PROCESS EVENT CREATOR AND FROM THE DYNAMIC EVENT CREATOR, IN ACCORDANCE WITH THE FILTER, AND SENDS THE EVENTS TO THE CONTROL STATION
678
THE CONTROL STATION ANALYZES EVENTS COLLECTED BY THE EVENT CONCENTRATOR AS THE EVENTS ARE COLLECTED OR ELSE AFTER THE EVENTS ARE COLLECTED
679
END
680

FIG. 26A

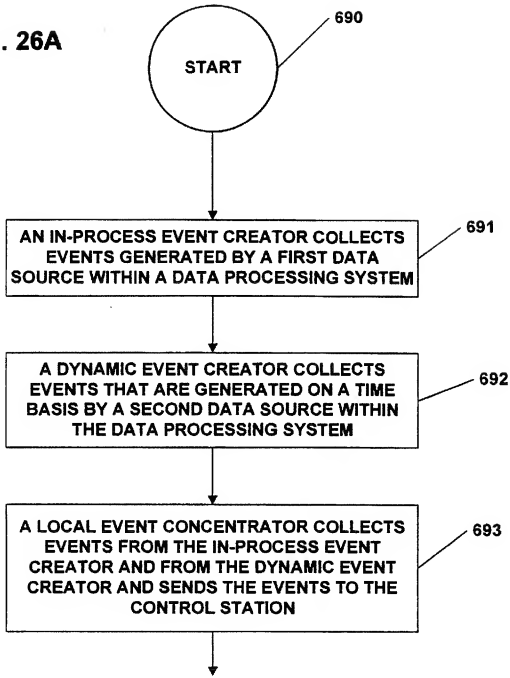

START
690
AN IN-PROCESS EVENT CREATOR COLLECTS EVENTS GENERATED BY A FIRST DATA SOURCE WITHIN A DATA PROCESSING SYSTEM
691
A DYNAMIC EVENT CREATOR COLLECTS EVENTS THAT ARE GENERATED ON A TIME BASIS BY A SECOND DATA SOURCE WITHIN THE DATA PROCESSING SYSTEM
692
A LOCAL EVENT CONCENTRATOR COLLECTS EVENTS FROM THE IN-PROCESS EVENT CREATOR AND FROM THE DYNAMIC EVENT CREATOR AND SENDS THE EVENTS TO THE CONTROL STATION
693

FIG. 26B

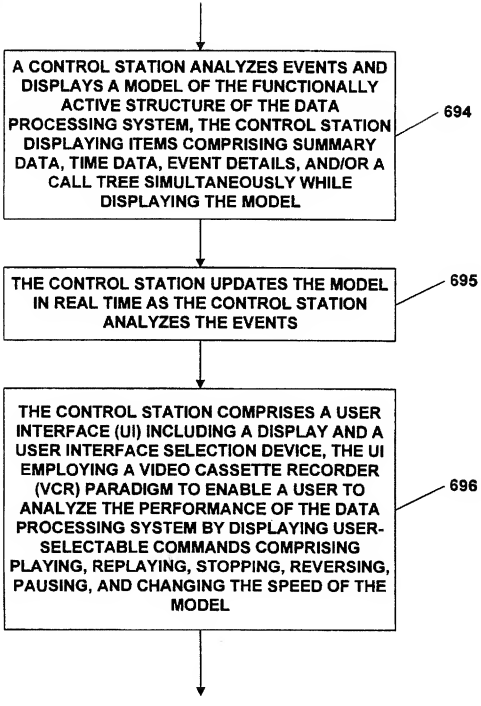

A CONTROL STATION ANALYZES EVENTS AND DISPLAYS A MODEL OF THE FUNCTIONALLY ACTIVE STRUCTURE OF THE DATA PROCESSING SYSTEM, THE CONTROL STATION DISPLAYING ITEMS COMPRISING SUMMARY DATA, TIME DATA, EVENT DETAILS, AND/OR A CALL TREE SIMULTANEOUSLY WHILE DISPLAYING THE MODEL
694
THE CONTROL STATION UPDATES THE MODEL IN REAL TIME AS THE CONTROL STATION ANALYZES THE EVENTS
695
THE CONTROL STATION COMPRISES A USER INTERFACE (UI) INCLUDING A DISPLAY AND A USER INTERFACE SELECTION DEVICE, THE UI EMPLOYING A VIDEO CASSETTE RECORDER (VCR) PARADIGM TO ENABLE A USER TO ANALYZE THE PERFORMANCE OF THE DATA PROCESSING SYSTEM BY DISPLAYING USER-SELECTABLE COMMANDS COMPRISING PLAYING, REPLAYING, STOPPING, REVERSING, PAUSING, AND CHANGING THE SPEED OF THE MODEL
696

FIG. 26C

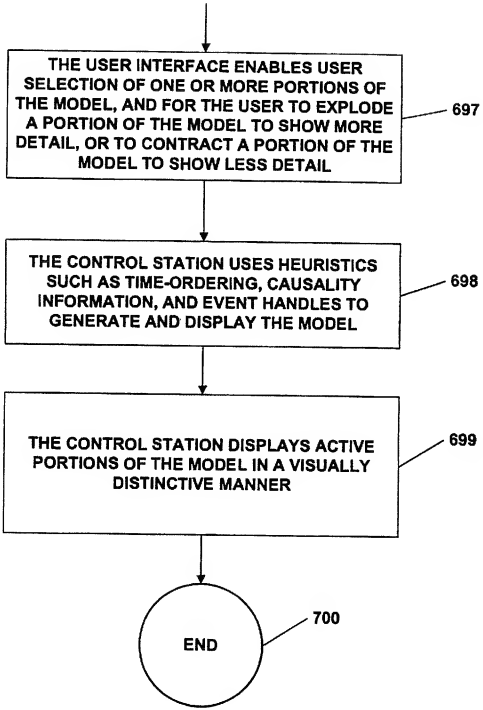

THE USER INTERFACE ENABLES USER SELECTION OF ONE OR MORE PORTIONS OF THE MODEL, AND FOR THE USER TO EXPLODE A PORTION OF THE MODEL TO SHOW MORE DETAIL, OR TO CONTRACT A PORTION OF THE MODEL TO SHOW LESS DETAIL
697
THE CONTROL STATION USES HEURISTICS SUCH AS TIME-ORDERING, CAUSALITY INFORMATION, AND EVENT HANDLES TO GENERATE AND DISPLAY THE MODEL
698
THE CONTROL STATION DISPLAYS ACTIVE PORTIONS OF THE MODEL IN A VISUALLY DISTINCTIVE MANNER
699
END
700

FIG. 27A

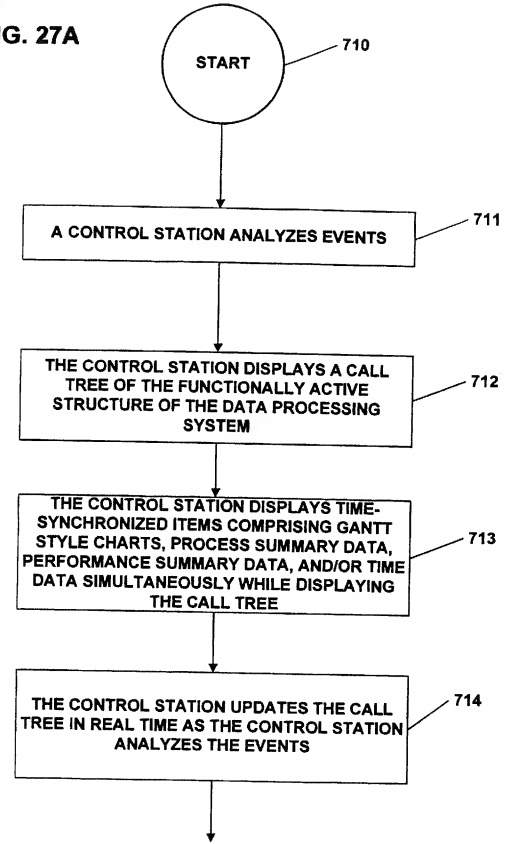

START
710
A CONTROL STATION ANALYZES EVENTS
711
THE CONTROL STATION DISPLAYS A CALL TREE OF THE FUNCTIONALLY ACTIVE STRUCTURE OF THE DATA PROCESSING SYSTEM
712
THE CONTROL STATION DISPLAYS TIME-SYNCHRONIZED ITEMS COMPRISING GANTT STYLE CHARTS, PROCESS SUMMARY DATA, PERFORMANCE SUMMARY DATA, AND/OR TIME DATA SIMULTANEOUSLY WHILE DISPLAYING THE CALL TREE
713
THE CONTROL STATION UPDATES THE CALL TREE IN REAL TIME AS THE CONTROL STATION ANALYZES THE EVENTS
714

FIG. 27B

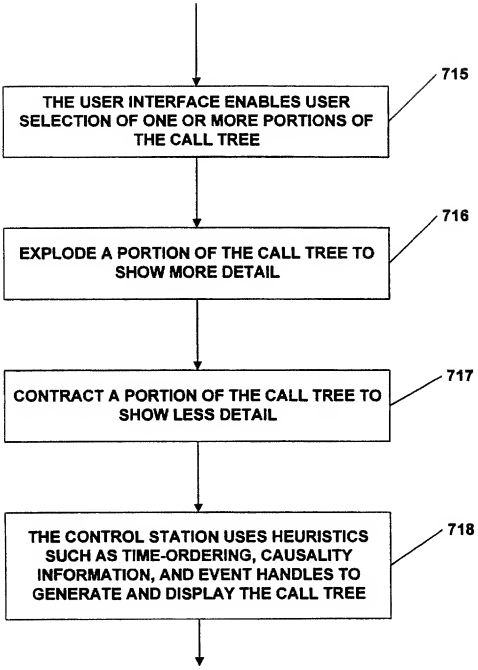
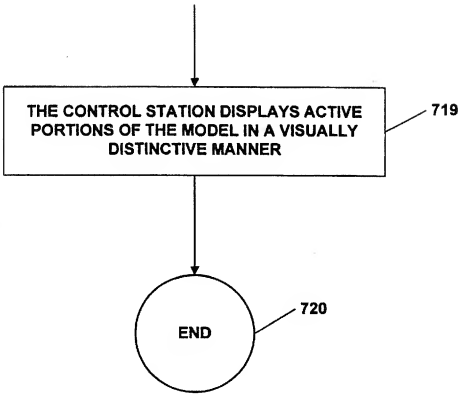

THE USER INTERFACE ENABLES USER SELECTION OF ONE OR MORE PORTIONS OF THE CALL TREE
715
EXPLODE A PORTION OF THE CALL TREE TO SHOW MORE DETAIL
716
CONTRACT A PORTION OF THE CALL TREE TO SHOW LESS DETAIL
717
THE CONTROL STATION USES HEURISTICS SUCH AS TIME-ORDERING, CAUSALITY INFORMATION, AND EVENT HANDLES TO GENERATE AND DISPLAY THE CALL TREE
718

FIG. 27C


THE CONTROL STATION DISPLAYS ACTIVE PORTIONS OF THE MODEL IN A VISUALLY DISTINCTIVE MANNER
719
END
720

METHOD AND APPARATUS FOR ANALYZING PERFORMANCE OF DATA PROCESSING SYSTEM

TECHNICAL FIELD

This invention relates generally to data processing and, more particularly, to a method and apparatus for analyzing the performance of a data processing system.

COPYRIGHT NOTICE/PERMISSION

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the U.S. Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever. The following notice applies to the software and data as described below and in the drawings hereto: Copyright © 1997–1999, Microsoft Corporation, All Rights Reserved.

BACKGROUND OF THE INVENTION

In the field of data processing it is a well known problem that software developers usually require a period of time to identify and resolve functional and performance issues in the code they have written or integrated. There can be many reasons for such issues, including the basic system and software architecture; non-optimized and/or flawed coding; the choice of, utilization of, and contention for system resources; timing and synchronization; system loading; and so forth.

Particularly in the area of distributed computer networks, it can be extremely difficult for software developers to observe and isolate undesirable system performance and behavior. A distributed computer network is defined herein to mean, at a minimum, a data processing system that utilizes more than one software application simultaneously or that comprises more than one processor.

For example, a single box or machine which is running two or more processes, such as a data base application and a spreadsheet application simultaneously, fulfills this definition. Also, a single article such as a hand-held computer may comprise more than one microprocessor and thus fulfills the definition.

More commonly, however, distributed computer networks may comprise two or more physical boxes or machines, often hundreds or even millions (in the case of the Internet). A software developer trying to monitor and analyze the operation and behavior of such complex computer networks is faced with a very daunting task.

For example, a developer may be writing or have written a server component that performs credit checks. This software component is used in a larger application that performs order entry processing. There are several other server components in the system (such as inventory verification, order validation, etc.) some of which run on the same server and some which run on a separate server (where the inventory database resides). To complicate matters, each component could reside on a computer system in a different state or country. If the application is not performing or behaving well, the developer needs to figure out if there is a performance or behavioral problem and, if so, be able to determine exactly where the trouble spots are.

In the prior art the developer had to modify his or her application, by writing trace statements in the code and having the application write to a log file what was going on at different places in the network. Then all of the log files would need to be collected, merged, and sorted. The developer would then have to sift through the data in a time-intensive fashion and attempt to determine the performance problem.

There are several serious deficiencies with the prior approach.

One problem is that only instrumented code can be analyzed. That means source code must be modified, recompiled, and re-deployed. This is a serious issue with the widespread use of operating system services and component technology in today's applications. Users are typically unable to recompile operating system and third party components, because they do not have physical or legal access to the source code. When they do have access to the source code, they are still unable to instrument them effectively, because they do not understand the component source code that they do have.

Another problem is that the modifications to code made by developers in an attempt to analyze its performance themselves adversely impact the application's performance. Further, the development of a highly efficient mechanism for recording the application data is non-trivial. Typical implementations involve writing data to disk. Even if the input/output (I/O) is buffered asynchronously, it can have an adverse impact on the application being monitored (e.g. masking actual application I/O).

A further problem is that understanding control flow during transitions is very hard. Typically, in a large distributed application, transitions to separate processes, or to processes running on separate machines, are common, and may happen simultaneously. Since events have to be manually merged by the developer, it is typically hard to determine which suspension in one process corresponds to resumption in another.

An additional problem is that frequently there are a large number of application areas that might need to be analyzed; however, not all of them may need to be analyzed at the same time. Developers who manually instrument their code must incorporate a selection technology to enable different portions to be analyzed. Otherwise, the load of all of the instrumentation has a severe impact on the analysis. This also requires a complex mechanism for developers to specify which information to collect on which machine.

Yet another problem is that for distributed applications, logs from multiple machines (and often multiple logs per machine) must be merged and sorted. Without synchronized clocks, this task is very difficult. As well, if the log files are in different formats (which is likely if they are from different developers or companies), then the data must be translated into common formats.

The result of all the effort described in this section is a very long list of analysis data. Manually analyzing and isolating performance problems from this amount of data is a very complex and difficult task.

One further problem with known performance analysis of data processing systems is that very often such analysis provides opportunities for breaching the data security of such systems.

There exists known performance monitoring software in various forms. Among them is software known as PerfMon software, which is commercially available from Microsoft Corporation. PerfMon software is a utility which, among other things, can provide an indication of the utilization of the computer's central processor unit (CPU) and memory

unit. PerfMon software operates by sampling. That is, it tracks continuous data by monitoring a machine and looking at its behavior. It can track the free space on a disk, monitor network usage, and so on, but it cannot gather event-based information, such as what function was most recently started.

There also exist known tools called profilers. These look at a single executing software application and try to understand its performance. They do this either by monitoring the program (in a similar way to PerfMon software), or else they hook into the program they are monitoring and generate "events" each time a program subcomponent (function) commences or completes. Profilers typically have a massive impact on the performance and behavior of an application, because they are intrusive, and they typically require special compiler support. Their data is so detailed that it is normally impractical to use them, particularly in a distributed computing environment such as the one described above.

The Windows NT® PerfMon utility, commercially available from Microsoft Corporation, provides an extensible architecture for the collection and display of arbitrary application and system counters and metrics. Windows NT provides base counters for the system for the purpose of monitoring CPU and memory utilization. It also provides counters for networks, disks, devices, processes, and so forth. Most system objects export counters. Many applications available from Microsoft Corporation (such as MTS and SQL Server) and other suppliers provide additional counters.

Therefore, there is a substantial need to provide software developers with automated tools for efficiently analyzing the performance, function, and behavior of their applications.

There is also a substantial need to provide such developers with tools for analyzing the performance, function, and behavior of their applications, either while the applications are executing or post mortem, and without significantly affecting the performance or data security characteristics of the applications

In addition, there is a substantial need, in a commercial environment, to provide Application Program Interfaces (APIs) to such tools.

SUMMARY OF THE INVENTION

The above-mentioned shortcomings, disadvantages and problems are addressed by the present invention, which will be understood by reading and studying the Detailed Description of the Invention. However, a brief summary of the invention will first be provided.

The present invention includes a number of different aspects for analyzing the performance of a data processing system. For the purposes of describing this invention, the term "performance" is intended to include within its meaning not only the operational performance, but also the function, structure, operation, and behavior of a data processing system.

While the invention has utility in analyzing the performance of a software application that is executing on a distributed data processing system, its utility is not limited to such, and it has utility in analyzing the performance of computer hardware, computer software of all types including data structures, and a wide spectrum of data processing systems comprising both computer hardware and computer software.

Insofar as the overall architecture and operation of the present invention is concerned, each machine where a portion of a distributed software application executes has at least one local event concentrator (LEC). In addition, there is at least one in-process event creator (IEC) and at least one dynamic event creator (DEC) per machine. The function of an IEC is to monitor the executing process for particular situations that occur which the developer wants to be monitored and to create an "event" that can be captured and later analyzed. The function of a DEC is similar to that of an IEC, but it monitors some aspect of the system operation that the developer wants to be monitored on a periodic or time basis and creates an "event" that can also be captured and later analyzed.

The developer can specify by means of a "filter" what to look for in the system under examination. This narrows the scope of the search to what is of interest to the developer and reduces the burden on the performance monitoring system.

When the IEC and DEC create events, they send them to the LEC, which collects them and temporarily stores them, either until the developer requests them or a developer-defined condition or "trigger" occurs, whereupon the LEC sends the events to the developer's control station. The control station analyzes the events and visually displays the results of the analysis to the developer in a multi-windowed, time-synchronized display.

In order to prevent the collection of information from adversely affecting the performance of the system, the IEC and DEC are only active when they are carrying out the developer's orders to monitor certain things. Otherwise they are dormant and do not affect the performance. When an IEC is activated and is monitoring process execution for particular situations, it creates a stream of events during "normal" execution and sends them to the LEC. However, the LEC doesn't send them through the network to the developer's control station until they are needed.

In another aspect of the invention, a data design structure allows two communicating entities to describe their interactions and inter-relationships despite knowing almost nothing about each other. The data design structure includes pre-defined event fields and custom fields, and it breaks up the application into a series of black boxes and maps out the entities of the network and their inter-relationships for displaying to the developer an animated model of the application as it is executing, either in real time or "post mortem".

In another aspect, the invention provides for user-defined triggers which cause the performance analysis software to passively buffer events until a malfunction occurs, then dump the buffered data and analyze it. This allows low-impact monitoring, since no information is stored until something of interest happens.

In another aspect, the invention comprises filter reduction features with which the developer can specify exactly what information within the network is of interest. Filter reduction is used to narrow the scope of the filter to extract only the information of interest and hence reduce the performance impact of monitoring.

In another aspect, the invention comprises filter combination features with which different users can specify individual filters that can be combined. The LEC can be multi-threaded and combine filters submitted by multiple users.

In another aspect, the invention comprises a filter user interface which is a graphical representation of the machines, entities, and events making up the network. The user can easily pick those of interest, using displayed lists and Boolean operator tabs, or can simply write an order in text format which is converted to the appropriate filter.

In another aspect, the invention comprises APIs for registration, in-process event creators, dynamic event creators, and other functions implementing the various aspects of the invention.

In another aspect, the invention provides for the automatic generation of an animated application model of the process under examination. A dynamic diagram of the application is automatically displayed as the various constituents interact. A video cassette recorder (VCR) paradigm is used to "play, replay, stop, pause, change speed, and reverse" the display, to enable the user to see what's happening as the application executes.

In another aspect, the invention provides for automatic, synchronized display of all performance analysis data. A number of user-customized, synchronized display windows show the constituent parts of the application execution and the corresponding performance characteristics, in both Gantt chart and graphical modes, either in real-time or post-mortem. A timeline window displays a visual representation of the timing of all related events. A summary window displays a distillation of the system performance during a user-selected time slice.

In another aspect, the invention provides suitable data security mechanisms throughout the network being monitored. Discretionary access is applied to the collection of data from a specific machine.

The present invention describes systems, clients, servers, methods, and computer-readable media of varying scope. In addition to the aspects and advantages of the present invention described in this summary, further aspects and advantages of the invention will become apparent by reference to the drawings and by reading the Detailed Description that follows.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention is pointed out with particularity in the appended claims. However, other features of the invention will become more apparent and the invention will be best understood by referring to the following Detailed Description in conjunction with the accompanying drawings in which:

FIG. **1** illustrates a hardware and operating environment in conjunction with which embodiments of the invention can be practiced;

FIG. **2** illustrates a system-level overview of an exemplary embodiment of the invention;

FIG. **3** illustrates a machine-level overview of an exemplary embodiment of the invention;

FIG. **4** illustrates in schematic fashion pre-defined event fields and custom fields, which are included in an event packet within an exemplary embodiment of the invention;

FIG. **5** illustrates a transition between two entities within the hardware and operating environment;

FIG. **6** is a table which illustrates how pre-defined event fields are used to establish a relationship between a source and a target entity;

FIG. **7** illustrates in schematic fashion how events selected by a user are monitored.

FIG. **8** illustrates a process of filter reduction as used within an exemplary embodiment of the invention;

FIG. **9** illustrates a process of filter combination as used within an exemplary embodiment of the invention;

FIG. **10** illustrates another process of filter combination as used within an exemplary embodiment of the invention;

FIG. **11** illustrates a screen print of an exemplary user interface for specifying a filter;

FIG. **12** illustrates a system level overview of an exemplary embodiment showing where APIs of the present invention can appear within the software architecture of a distributed computing system;

FIG. **13** illustrates a screen print of an animated application model which the present invention generates to show the structure and activity of an application whose performance is being studied;

FIG. **14** illustrates various user interface features of an animated application model in an exemplary embodiment of the invention;

FIG. **15** illustrates a representative display of performance data in an exemplary embodiment of the invention;

FIG. **16** illustrates a screen print of an exemplary display of performance data;

FIG. **17** illustrates screen print of a timeline display of performance data;

FIG. **18** illustrates a screen print of summary display of performance data;

FIG. **19** illustrates a screen print of several synchronized sets of performance data;

FIG. **20** A–C is a flowchart of a method illustrating an exemplary embodiment of overall data collection architecture and how data is collected via the IECs, DECs, and LECs;

FIG. **21** A–B is a flowchart of a method illustrating an exemplary embodiment of overall data design and how the VSA determines and maps relationships between entities;

FIG. **22** A–B is a flowchart of a method illustrating an exemplary embodiment of triggers;

FIG. **23** A–B is a flowchart of a method illustrating an exemplary embodiment of filter reduction;

FIG. **24** A–B is a flowchart of a method illustrating an exemplary embodiment of filter combination;

FIG. **25** A–B is a flowchart of a method illustrating an exemplary embodiment of a user interface for specifying one or more filters;

FIG. **26** A–C is a flowchart of a method illustrating an exemplary embodiment of automatic generation of an animated application model; and

FIG. **27** A–C is a flowchart of a method illustrating an exemplary embodiment of a user interface for displaying the performance analysis of the system under examination.

DETAILED DESCRIPTION OF THE INVENTION

In the following Detailed Description of exemplary embodiments of the invention, reference is made to the accompanying drawings that form a part hereof, and which show by way of illustration specific exemplary embodiments in which the invention can be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention. It is to be understood that other embodiments can be utilized and that logical, mechanical, electrical, and other changes can be made without departing from the spirit and scope of the present invention. The following Detailed Description is, therefore, not to be taken in a limiting sense, and the scope of the present invention is defined only by the appended claims.

The Detailed Description is divided into six sections. In the first section, a Glossary of Terms is provided. In the second section, an Exemplary Hardware and Operating

Environment in conjunction with which embodiments of the invention can be practiced is described. In the third section, a System Level Overview of the invention is presented. In the fourth section, Exemplary Embodiments of the Invention are provided. In the fifth section, Methods of Exemplary Embodiments of the Invention are provided. Finally, in the sixth section, a Conclusion of the Detailed Description is provided.

Glossary of Terms

The following section provides definitions of various terms used in the Detailed Description:

ADO—ActiveX® Data Objects, a high-level programming interface from Microsoft Corporation for data objects which can be used to access different types of data, including web pages, spreadsheets, and other types of documents. It is designed to provide a consistent way of accessing data regardless of how the data is structured.

API—Application Program Interface, a language and message format used by an application program to communicate with the operating system, middleware, or other system program such as a database management system. APIs are generally implemented by writing function calls in the application program, which provide the linkage to a specific subroutine for execution. Operating environments typically provide an API so that programmers can write applications consistent with the operating environment.

COM—Component Object Model, a component software architecture from Microsoft Corporation which defines a structure for building program routines or objects that can be called up and executed in a Microsoft Windows® operating system environment.

DCOM—Distributed Component Object Model, developed by Microsoft Corporation, it is an extension of the Component Object Model (COM), which enables object-oriented processes distributed across a network to communicate with one another.

Entity—a functional component in a data processing system, such as a client, server, or data source.

GUID—a Globally Unique Identifier within a data processing system. Within the present invention it is used to identify, for example, a COM object, an event source, an event, an event category, and any other system object that requires guaranteed unique identification from multiple independent generators.

Machine—a minimal data processing system comprising at least a processor and a memory, the processor executing software instructions which are stored in the memory.

Middleware—a category of processes between the application itself and backend processes such as databases, network connections, and so forth. Applications that run on currently available operating systems typically require services above and beyond those provided by the operating system. These services are often no longer written by the application developer but by a third party (which can be the operating system vendor). The term "middleware" indicates the position of these common services within the software architecture relative to the application.

MTS—Microsoft Transaction Server (MTS), a feature of the Microsoft Windows NT Server® operating system that facilitates the development and deployment of server-centric applications built using Microsoft's Component Object Model (COM) technologies.

NTS—Windows NT Server®, a version of the Microsoft Windows® operating system. There are currently two commercially available versions of Windows NT: Windows NT Server®, designed to act as a server in networks, and Windows NT Workstation® for stand-alone or client workstations.

PerfMon—Performance Monitor, a utility provided with Microsoft Corporation's Windows NT® operating system which enables the performance monitoring of all services running on a system.

RPC—Remote Procedure Call, a programming interface that allows a program on one computer to execute a program on a server computer. Using RPC, a system developer need not develop specific procedures for the server. The client program sends a message to the server with appropriate arguments, and the server returns a message containing the results of the program executed.

Windows® operating system—an operating system commercially available from Microsoft Corporation for several different computing platforms.

Exemplary Hardware and Operating Environment

FIG. **1** illustrates a hardware and operating environment in conjunction with which embodiments of the invention can be practiced. The description of FIG. **1** is intended to provide a brief, general description of suitable computer hardware and a suitable computing environment with which the invention can be implemented. Although not required, the invention is described in the general context of computer-executable instructions, such as program modules, being executed by a computer, such as a personal computer (PC). This is one embodiment of many different computer configurations, some including specialized hardware circuits to analyze performance, that can be used to implement the present invention. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types.

Moreover, those skilled in the art will appreciate that the invention can be practiced with other computer-system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network personal computers ("PCs"), minicomputers, mainframe computers, and the like. The invention can also be practiced in distributed computing environments where tasks are performed by remote processing devices linked through a communications network. In a distributed computing environment, program modules can be located in both local and remote memory storage devices.

FIG. **1** shows a general-purpose computing or information-handling system **80**. This embodiment includes a general purpose computing device such as personal computer (PC) **20**, that includes processing unit **21**, a system memory **22**, and a system bus **23** that operatively couples the system memory **22** and other system components to processing unit **21**. There may be only one or there may be more than one processing unit **21**, such that the processor computer **20** comprises a single central-processing unit (CPU), or a plurality of processing units, commonly referred to as a parallel processing environment. The computer **20** can be a conventional computer, a distributed computer, or any other type of computer; the invention is not so limited.

In other embodiments other configurations are used in PC **20**. System bus **23** can be any of several types, including a memory bus or memory controller, a peripheral bus, and a local bus, and can use any of a variety of bus architectures. The system memory **22** may also be referred to as simply the memory, and it includes read-only memory (ROM) **24** and

random-access memory (RAM) **25**. A basic input/output system (BIOS) **26**, stored in ROM **24**, contains the basic routines that transfer information between components of personal computer **20**. BIOS **26** also contains start-up routines for the system.

Personal computer **20** further includes hard disk drive **27** having one or more magnetic hard disks (not shown) onto which data is stored and retrieved for reading from and writing to hard-disk-drive interface **32**, magnetic disk drive **28** for reading from and writing to a removable magnetic disk **29**, and optical disk drive **30** for reading from and/or writing to a removable optical disk **31** such as a CD-ROM, DVD or other optical medium. Hard disk drive **27**, magnetic disk drive **28**, and optical disk drive **30** are connected to system bus **23** by a hard-disk drive interface **32**, a magnetic-disk drive interface **33**, and an optical-drive interface **34**, respectively. The drives **27**, **28**, and **30** and their associated computer-readable media **29**, **31** provide nonvolatile storage of computer-readable instructions, data structures, program modules and other data for personal computer **20**. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk **29** and a removable optical disk **31**, those skilled in the art will appreciate that other types of computer-readable media which can store data accessible by a computer can also be used in the exemplary operating environment. Such media may include magnetic tape cassettes, flash-memory cards, digital video disks (DVD), Bernoulli cartridges, RAMs, ROMs, and the like.

In various embodiments, program modules are stored on the hard disk drive **27**, magnetic disk **29**, optical disk **31**, ROM **24** and/or RAM **25** and can be moved among these devices, e.g., from hard disk drive **27** to RAM **25**. Program modules include operating system **35**, one or more application programs **36**, other program modules **37**, and/or program data **38**. A user can enter commands and information into personal computer **20** through input devices such as a keyboard **40** and a pointing device **42**. Other input devices (not shown) for various embodiments include one or more devices selected from a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit **21** through a serial-port interface **46** coupled to system bus **23**, but in other embodiments they are connected through other interfaces not shown in FIG. **1**, such as a parallel port, a game port, or a universal serial bus (USB) interface. A monitor **47** or other display device also connects to system bus **23** via an interface such as a video adapter **48**. In some embodiments, one or more speakers **57** or other audio output transducers are driven by sound adapter **56** connected to system bus **23**. In some embodiments, in addition to the monitor **47**, system **80** includes other peripheral output devices (not shown) such as a printer or the like.

In some embodiments, personal computer **20** operates in a networked environment using logical connections to one or more remote computers such as remote computer **49**. Remote computer **49** can be another personal computer, a server, a router, a network PC, a peer device, or other common network node. Remote computer **49** typically includes many or all of the components described above in connection with personal computer **20**; however, only a storage device **50** is illustrated in FIG. **1**. The logical connections depicted in FIG. **1** include local-area network (LAN) **51** and a wide-area network (WAN) **52**, both of which are shown connecting PC **20** to remote computer **49**; typical embodiments would only include one or the other. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When placed in a LAN networking environment, PC **20** connects to local network **51** through a network interface or adapter **53**. When used in a WAN networking environment such as the Internet, PC **20** typically includes modem **54** or other means for establishing communications over network **52**. Modem **54** may be internal or external to PC **20** and connects to system bus **23** via serial-port interface **46** in the embodiment shown. In a networked environment, program modules depicted as residing within PC **20** or portions thereof may be stored in remote-storage device **50**. Of course, the network connections shown are illustrative, and other means of establishing a communications link between the computers can be substituted.

Software can be designed using many different methods, including object-oriented programming methods. C++ and Java are two examples of common object-oriented computer programming languages that provide functionality associated with object-oriented programming. Object-oriented programming methods provide a means to encapsulate data members (variables) and member functions (methods) that operate on that data into a single entity called a class. Object-oriented programming methods also provide a means to create new classes based on existing classes.

An object is an instance of a class. The data members of an object are attributes that are stored inside the computer memory, and the methods are executable computer code that act upon this data, along with potentially providing other services. The notion of an object is exploited in the present invention in that certain aspects of the invention are implemented as objects in some embodiments.

An interface is a group of related functions that are organized into a named unit. Some identifier can uniquely identify each interface. Interfaces have no instantiation; that is, an interface is a definition only without the executable code needed to implement the methods that are specified by the interface. An object can support an interface by providing executable code for the methods specified by the interface. The executable code supplied by the object must comply with the definitions specified by the interface. The object can also provide additional methods. Those skilled in the art will recognize that interfaces are not limited to use in or by an object-oriented programming environment.

System Level Overview

FIG. **2** illustrates a system-level overview of an exemplary implementation of the invention. The invention has utility in the area of data processing, where it can be used to analyze the performance of a data processing system, and in particular application software, whether under development, undergoing testing, or in full utilization. The invention is commercially available from Microsoft Corporation as the "Visual Studio"® development system or "Visual Studio Analyzer"®. In addition, certain portions of the invention are provided within the Microsoft Windows® operating system.

The "Visual Studio" development system collects application data by use of instrumentation within the application environment in an efficient, distributed collection architecture. Any application built with any development tool can be automatically analyzed and diagnosed, provided it uses standard middleware and operating system components. There is no requirement for any changes to the application itself.

As mentioned in the Background section earlier, distributed data processing systems can be relatively simple or extremely complex. The developer of software operating on

a distributed data processing system is usually faced with serious challenges in understanding the functional operation and behavior of such software as it is executing.

The system illustrated in FIG. 2 is a globally distributed system in which different machines **100**, **102**, **104**, **106**, and **108** are physically located on several different continents. These machines are shown as interconnected via hardwire, fiber-optic cable, radio frequency, or other suitable links **130**, **132**, **134**, and **136** in an arbitrary network arrangement spanning a large portion of the globe. The difficulties in understanding and trouble-shooting systems of this complexity have been significant until the present invention.

The present invention enables complex distributed applications to be readily understood and analyzed, notwithstanding that the machines on which they are running may be thousands of miles apart, and notwithstanding that the developer may not have access to source code for the underlying software upon which his or her application is running.

With reference to FIG. **2**, the box identified as VSA **100** is a control and display station that comprises computer hardware and software. VSA **100** is coupled to one or more machines, e.g. machines **102**, **104**, **106**, and **108**. Each machine includes a Local Event Concentrator (LEC) **112**, **152**. One LEC is provided per physical machine, although in a different implementation more could be provided if desired. VSA **100** activates an LEC when it wants that LEC to start collecting events, and VSA **100** deactivates an LEC when it wants it to stop collecting events. In addition to VSA **100**, other client machines can also activate or deactivate an LEC **112** or **152**.

Each LEC **112**, **152** is coupled to a respective process space **110**, **150**. Each process space **110**, **150** can each comprise a group of In-process Event Creators (IECs), such as IECs #1.1 through #1.N in group **110**.

Each LEC **112**, **152** is further coupled to a respective process space **114**, **154**. Each process space **114**, **154** can each comprise a group of Dynamic Event Creators (DECs), such as DECs #1.1 through #1.N in group **114**. Process spaces **110** and **114** can be identical or different for machine **104**; likewise for the process spaces **150**, **154** associated with machine **106**. While all DECs are shown in FIG. **2** as residing in process spaces **114**, **154**, in one embodiment DECs that capture global machine state (such as PerfMon data) reside only within the LEC process space.

Machine-Level Overview

FIG. **3** illustrates a machine-level overview of an exemplary embodiment of the invention. In FIG. **7** three major portions of the process space of a machine are shown in the form of Applications **190**, Operating System **191**, and Additional Components **192**.

In one aspect, the invention comprises one local event concentrator (LEC) **199** for each machine. Applications portion **190** has an IEC **193** associated with it; Operating Systems portion **191** has an IEC **195** associated with it; and Additional Components portion **192** has an IEC **197** associated with it.

There is at least one dynamic event creator (DEC) per machine, such as DEC **189**, which is in the process space of LEC **199**. It will be apparent to one of ordinary skill in the art that DECs could be provided for each portion **190**, **191**, **192** of the machine's process space. This is shown in FIG. **3** by DEC boxes **194**, **196**, **198** having dashed lines.

Events created by IECs **193**, **195**, **197** and DECs **189**, **194**, **196**, **198** are collected by LEC **199**. The LEC **199** collects events generated by the IECs and DECs and sends these events to the user's control station, VSA **100**, for analysis and display in a user-determined format.

IECs and DECs reside in the process space of data sources within a machine, and they "report on" these data sources. They each do this by creating events that are sent to and collected by the LEC. They are active only when the user is interested in knowing about these events and in understanding the system performance.

IECs and DECs differ in their purpose. An IEC creates an event when a user-specified condition (other than time-valued data) occurs. An example could be "a COM event in Machine A". A DEC, on the other hand, creates an event to reflect data whose value is measured on a periodic or time basis. An example could be Perfson data reflecting CPU utilization.

As mentioned in the Summary section above, the system described herein for analyzing the performance of a data processing system is a comprehensive one with many different aspects, each of which will now be described in the section below entitled Exemplary Embodiments of the Invention.

EXEMPLARY EMBODIMENTS OF THE INVENTION

Collection, Capture & Transmission of Data

Data collection begins in the IECs. An IEC is a subroutine that marshals the desired data into a special format and puts it in a shared memory buffer. As mentioned above, IECs reside in the process space of a data source.

An IEC exports two main functions: IsActive and FireEvent. The IsActive function is used by data sources to determine if any analysis is being performed against a particular data source. When a piece of code reaches a point of interest, the IsActive function is called, which returns True or False as to whether or not anyone is interested. If the IsActive status condition is set True for a particular data source, the FireEvent function is used to dispatch an event to the centralized collection system of the requesting user. If IsActive returns False, an entity can reduce any adverse performance impact by not formatting data for FireEvent. The FireEvent function is implemented in both a synchronous and an asynchronous manner in the present invention.

When an LEC has been activated by the VSA **100**, it can turn an IEC on or off, i.e. it switches its IsActive status to True or False. That Boolean status is maintained in the process, so there are really never any in-process transitions, and the code never changes. When IsActive is True, events are generated. When the VSA **100** user wants to stop monitoring events, everything can be quickly disconnected. IsActive is set to False, and the application never changes.

Also, when an LEC has been activated by VSA **100**, it can turn a DEC on or off, depending upon whether the DEC is to collect events. When a DEC is to stop collecting events, an LEC simply turns it off. As for IECs, an LEC starts and stops DECs as specified by a user-specified filter, as will be discussed further below.

Instead of turning individual IECs on and off, a portion of the IECs or all of the IECs can be turned on or off. The same applies to other structures of the invention, including DECs and LECs.

To improve system-wide efficiency, the operating system or middleware defers the creation of an IEC until the user actually begins collection of events. IECs are only created

for users who desire to monitor system performance. They are automatically created when needed. This ensures that, if the system is not under analysis, the performance impact of operating the performance analyzer is negligible. Additionally, the system is ahle to remove all of the IECs from memory when analysis completes, so that a system wherein analysis has finished behaves with the same characteristics as before performance began, unlike many traditional tools.

IECs and DECs are created hy the operating system, middleware, and application components that are sourcing the events. The creation of an IEC will now be described. Assume that a middleware entity wants to fire events. It asks the operating system to create an IEC. The operating system creates an IEC "reference", ready for the IEC in case the user wants to start monitoring data. When the user wants to start monitoring data, the LEC tells the operating system to convert the IEC "reference" into a real IEC. The operating system converts all the IEC references into real IECs the first time they are used.

Events from IECs in process spaces **110**, **150** are passed to a respective LEC **112**,**152** via shared memory buffers. This allows the event to be communicated without requiring a process context switch. Each IEC has its own buffer in shared memory, to ensure that conflicts between events and locking do not distort system performance.

In one currently implemented embodiment there is only one LEC per machine. It collects events from all IECs in all processes on the system that are being analyzed, and it sends the desired events back to the VSA **100**. Since this communication is likely to be cross-machine, an efficient batching mechanism is used to reduce network traffic, and transmission is scheduled for low-system load times. To ensure efficient dispatch of events across the network, the LEC process runs at a lower than normal priority. This means that events will tend to be flashed across the network when the machine is not busy running the real application or when the real application is blocked, e.g., when it is waiting for data to be read from disk. To further reduce performance impact, events from many IECs are collected together and will not be sent more than some fixed period of time, e.g. every one-half to one second in one embodiment. If the number of events to be sent exceeds the buffering capacity, events will either be sent immediately or thrown away, depending upon a setting made at the control station.

Communication between the VSA **100** and the LECs also exists to establish clock skews so that event times throughout the distributed application can be synchronized. Any known clock skew calculators can be used for this purpose.

A DEC is similar to an IEC except that it deals with data whose value can be measured continuously, and whose values need to be recorded at regularly scheduled intervals. To reduce system complexity and increase flexibility in handling data, these "measured" events are treated internally just like events that are triggered by the system's behavior. This allows collection, synchronization, and analysis of both event-driven and time-driven data.

As opposed to an IEC which reports on the occurrence of events (i.e. "this thing happened"), a DEC gathers information on a time basis, such as memory usage within the system, not necessarily events coming from within the application. For example, a DEC might every second measure the memory usage of the system and send back an event that says "current memory usage is 2 megabytes". A DEC could also report on disk usage or CPU usage. A DEC could be created within the application itself to measure application-specific parameters such as, for example, the number of queries currently executing within a database system or the number of words currently misspelled in a word-processing document. Generally speaking, a DEC can measure any continuously varying data, i.e., anything which could be represented by a graph.

The VSA **100** collects all reported information and stores it in an efficient centralized store. The centralized store can simply be a data file in which data is organized in a certain way, i.e. a memory-mapped file. Other embodiments of an efficient data store could be a relational database, an in-memory data structure, a regular file, or any other suitable structure which can handle large volumes of data with an efficient access time.

Once written to, it can be read many times. Data is organized so it's easy to write, since incoming data volume can be very high, and also so it's easy to read directly from disk, since dataset size will typically preclude loading all data into memory.

Since data collection for one embodiment of the invention doesn't involve a multiple update problem, this was taken into consideration in designing the data structure. File-mapped memory buffers were used so that information could be quickly retrieved from disk and stored into memory in an efficient way. Thus the system is able to receive potentially many thousands of events per second. It is stored on disk in the order that it arrives.

It will be apparent to one of ordinary skill in the art that the present invention is equally applicable to a distributed system in a single machine. A single machine can be running more than one process, for example an operating system and a data base application.

It will further be apparent to one of ordinary skill that if the performance cost of a context switch is not of great concern, then it could in fact be carried out, provided that one appropriately factors it into the performance analysis.

It will be appreciated that just because the LEC is collecting something doesn't mean that it is necessarily what the VSA user wants. As will be explained below, user-specified filtering can occur in the IEC or in the LEC to reduce the information. In addition, the LEC, in a currently implemented embodiment, can buffer all or a substantial portion of the information that it sends out to the VSA, so it sends bursts on the network rather than continuous traffic. In addition, it can also run as a lower priority, so it's buffering up all of the information rather than directly slowing down the application. In addition, it can further compress data to reduce network overhead.

Operation of VSA

The operation of the VSA will now be described. When an application starts up, the operating system software or the "middleware" that the application is using creates an IEC reference, and if there's an LEC on the system the IEC reference hooks itself up to the LEC. However, if no one is analyzing the system yet, there will be no LEC yet, and the IEC reference will remain unhooked up.

Then the IEC reference goes into quiescent mode. The application keeps running, and nothing special is going on to slow it down.

Now, if someone wants to analyze what's going on, they turn on the VSA **100**, and they indicate that they want to hook up to a particular machine, so it turns on an LEC on that system. That LEC connects to all of the IECs on that system, and it starts any DECs, for example to monitor CPU

usage. DECs measure and report on time-based interval events, while IECs watch for and report on the occurrence of events. It will be apparent to one of ordinary skill that while the LEC is created by the VSA **100** in a currently implemented embodiment, it could be automatically created when the first IEC reference exists.

The VSA user specifies what information is to be collected. That information is broken down per machine and passed to the LEC for each machine. The LEC then breaks that down, per IEC, and basically turns the IECs on or off where appropriate. When IsActive is set True in an IEC, it is no longer quiescent, and that IEC starts sending collected data to its associated LEC. When the user shuts down the VSA, the IECs, DECs, and LECs revert back to their quiescent states.

The interface between the VSA and an LEC can operate under DCOM. Everything else can run under COM, except for the shared memory communication between the IEC and the LEC. The IEC writes information into a shared memory buffer and never takes a process context switch. COM is used basically only for initialization.

A third party developer is able to write a COM interface for its application and use the VSA to analyze its performance. It doesn't have to link any additional libraries.

Data Design—Pre-Defined Event Fields and Custom Fields

FIG. **4** illustrates in schematic fashion pre-defined event fields and custom fields, which are included in an event packet within an exemplary embodiment of the invention. Pre-defined event fields are generally always present in an event packet, whether the user specifies them or not. Custom fields can also be assigned by a user. In the invention each event may include a number of pre-defined or standard pieces of information, as well as custom or arbitrary user-specified information. This information becomes important when filter reduction occurs, as will be described further below.

As shown in FIG. 4, a VSA event comprises pre-defined event fields **160** and custom fields **162**. Not all pre-defined event fields have to be provided for every event. Pre-defined event fields **160** enable the data structure of the invention. If the user doesn't specify pre-defined event fields, intelligent default values are automatically provided for them.

Custom fields **162** can be generated by the user, but none of them is essential to the data design.

What distinguishes pre-defined event fields from custom fields is that pre-defined event fields have pre-defined semantics and are therefore useable by the analysis mechanism to determine the interrelationship among events. Without pre-defined event fields, the analysis mechanism would be unable to make any reasonable deductions about the events and would only be able to provide a useless list of events. Further, the set of pre-defined event fields is optimized for effective and efficient analysis. The specific names and functions are described in Table 1 below.

Some important pre-defined event fields are the Machine, Process, Entity (referred to as "Component" in Table 1 below and in the APIs), Instance (referred to as "Session" in Table 1 below and in the APIs), and Handle fields, both for the Source as well as for the Target. Their use will be explained in greater detail below.

Pre-defined event fields are listed in Table 1 below:

TABLE 1

Pre-Defined Event Fields
Arguments
CausalityID
CorrelationID
DynamicEvent Data
Exception
ReturnValue
SecurityIdentity
SourceComponent
SourceHandle
SourceMachine
SourceProcess
SourceProcessName
SourceSession
SourceThread
TargetComponent
TargetHandle
TargetMachine
TargetProcess
TargetProcessName
TargetSession
TargetThread
Time
Entity
Instance

Because the default set of events is large, pre-defined event categories are provided to visually organize the events in the filter editor. Each event belongs to exactly one category, and each category may have any number of different events. Each category may also have any number of child categories. The combination of all of the events and categories makes a tree where the leaves are events and the branches are categories. Event categories have no semantic impact on the event but do allow the filter to be displayed, stored, and processed more efficiently. Event categories have merely an organizational function, in that they help the user understand events.

Pre-defined event categories are listed in Table 2 below:

TABLE 2

Pre-Defined Event Categories
All
Call/Return
Measured
Query/Result
Start/Stop
Transaction

Each event has a type. The type is used to distinguish events that come from DECs. The event type is also used to distinguish events that are outbound (CALL or ENTER) from those that are inbound (LEAVE or RETURN). This distinction is important to matching up the steps of four events mentioned later regarding a CALL/ENTER/LEAVE/RETURN sequence. If an event belongs to either of these categories, then it is called generic.

Event types are unrelated to event categories. Events of the same type may be in different categories, and, conversely, events in the same categories may be of different types.

There are different types of events. The event type is used to specify how VSA **100** should interpret the event. Event types are listed in Table 3 below:

TABLE 3

Event Types
Begin/End - correspond to a set of events that surround an action. Default - for a default event (or unspecified event type). Generic - for a simple event (not a grouped event). Measured - for DEC events. Outbound/Inbound - for call/return events. Outbound means the transition is "out" of the component. Inbound means the transition is "into" the component.

The data design of the present invention allows the user to define his or her own events and event taxonomy. However, to provide some basic interoperability between data (so that generic analysis tools can be written and/or used), in one embodiment of the invention some typical events are defined. Compliant event generators within this embodiment are encouraged to use these events rather than to define their own. This helps simplify the filter editor. Alternative embodiments could either have no typical events or a very large set of typical events. The choice of typical events is merely dictated by the kind of events that are expected to be common within the embodiment of the invention which is implemented.

Table 4 below identifies pre-defined events and their categories and types:

TABLE 4

Pre-Defined Events and Categories

Event	Category	Type
Call	Call/Return	Outbound
Call Data	Call/Return	Outbound
Component Start	Start/Stop	Begin
Component Stop	Start/Stop	End
Enter	Call/Return	Inbound
Enter Data	Call/Return	Inbound
Events Lost	Transaction	Generic
Leave Data	Call/Return	Outbound
Leave Exception	Call/Return	Outbound
Leave Normal	Call/Return	Outbound
Query Enter	Query/Result	Inbound
Query Leave	Query/Result	Outbound
Query Result	Query/Result	Inbound
Query Send	Query/Result	Outbound
Return	Call/Return	Inbound
Return Data	Call/Return	Inbound
Return Exception	Call/Return	Inbound
Return Normal	Call/Return	Inbound
Transaction Commit	Transaction	End
Transaction Rollback	Transaction	End
Transaction Start	Transaction	Begin
User	All	Generic

In Table 4, the "Category" descriptors are merely annotational, not semantic.

A brief description of each Event listed in the "Event" column will now be given:

A "Call" event is the first step of a four-part Call/Enter/Leave/Return transition. A function call is departing from a caller.

"Call Data" means subsidiary data to a call is being transmitted. This always follows a Call.

"Component Start" means a component has been created and is starting to execute (note that "component" in this sense is not the same as an "entity" as used herein; it means a real component).

"Component Stop" means a component has been destroyed and is stopping its execution (note the comment above).

"Enter" means the second step in a four-step transition. A function call is arriving at the callee.

"Enter Data" means subsidiary data to an Enter has been received.

"Events Lost" means the system has had to discard events to avoid overloading the eventing infrastructure.

"Leave Data" means subsidiary data to a leave has been transmitted from a callee to the caller.

"Leave Exception" means an exception (error) has been transmitted from the callee to the caller. This is the third step in the four-part transition.

"Leave Normal" means a success has been transmitted from the callee to the caller. This is the third step in the four-part transition.

"Query Enter" means a database query has arrived at the database.

"Query Leave" means a database query has been completed.

"Query Result" means a database query result set has started transmitting back to the caller.

"Query Send" means a database query has left the caller.

"Return" means the fourth step in the four-part transition. Control has returned to the caller.

"Return Data" means subsidiary data to a Return has been received at the caller.

"Return Exception" means an exception (error) has been received at the caller. This is the fourth step in the four-part transition.

"Return Normal" means a success has been received at the caller. This is the fourth step in the four-part transition.

"Transaction Commit" means a transaction has been committed successfully.

"Transaction Rollback" means a transaction was aborted.

"Transaction Start" means a new transaction was created and started.

"User" means an unknown event.

Data Design—E0/E1 Entity Transition

FIG. 5 illustrates a transition between two entities, E0 and E1, within the hardware and operating environment. A "transition" occurs when one entity (e.g. a program, process, or object) turns execution over to another to complete a specific task. In FIG. 5 the illustrated transition comprises four events, a Call event, an Enter event, a Leave event, and a Return event.

When understanding the structure and behavior of distributed systems, understanding transitions between different applications entities is important. The VSA employs an innovative data design that allows two communicating entities to describe their interactions despite knowing almost nothing about each other. Each participant in a transition provides only information about its environment, plus a unique identifier that allows the entity at the other end of the transition to link the pair of events. Every destination called needs to have a unique i.d., and every source of a Call has a unique i.d. In an embodiment which was implemented, these unique i.d.'s are GUIDs.

This design has a number of benefits. First, because entity systems typically already include a quasi-unique identifier

for transitions, no extra information needs to be transmitted between the two entities. Second, each entity data load is reduced through less duplicated data.

Each application is treated as a series of black boxes. A "transition" is defined as when an application moves from one of those boxes to another one. So if we have a Client and a Server, a transition occurs when we go to the Server, and another occurs when we go back. In a three-tier design, a transition occurs for Client to Server, Server to Database, Database to Server, and Server to Client movements. These are entity to entity transitions and not necessarily machine to machine transitions.

One example of an entity to entity transition is one COM client component calling a COM server component. Essentially four events represent that transition, which can be a remote procedure Call (RPC) within a distributed system. An event from the client says "I'm initiating a Call". An event at the server says "I've entered the server". An event at the server says "I'm leaving the server". And finally an event at the client says "I've returned". In the case of COM, an event occurs at both sides of the transition.

By looking at all or nearly all of these events and taking appropriate pieces of information about them and correlating them, a great deal of information is derived about the structure and performance of the system, and accordingly a performance model of the system can be constructed.

Data Design—Determination of Source/Target Relationship

FIG. **6** is a table which illustrates how pre-defined event fields are used to establish a relationship between a source entity and a target entity.

For each of the events involved in a Call, Enter, Leave, and Return sequence, the event producer specifies the Machine of the source, the Process of the source, the Entity (e.g. class, such as ADO) of the source, and the Instance of the source.

Thus the VSA knows the Machine, Process, Entity, and Instance at the Source for a Call event, but it doesn't know the Machine, Process, Entity, and Instance at the Target for a Call event. And for the Enter event, the situation is reversed. The VSA doesn't know it for the Source, but it does know it for the Target. In almost all cases the events are fired at the place the event is happening.

Using this information the VSA is able to piece together a functional block diagram of the system as described below.

There are basically two kinds of users that use VSA. There are people who give us events, and there is the actual end user who is collecting data to understand it. The data design of the invention is manipulated and used by the portion of the operating system that gives us events, and the end user doesn't really need to understand it in great depth. This format makes it possible to draw a block diagram of the system, even though no one piece knows what the system should look like.

In most existing systems, E0 and E1 have a very weak relationship. The data design of the present invention is innovative in that it can tolerate this weak relationship and still provide useful results. E0 doesn't really need to know what machine E1 is on, and vice versa. Even though these two entities communicate through the system, e.g. via COM, they don't really know about each other. So when a Call event is fired by E0, it doesn't really know whom it's talking to. When E1 fires the Enter event that goes with that Call event, it doesn't really know that that Enter event goes with that Call event. So the small amount of information that the operating system has is leveraged to make sure that the Call event maps the Enter event. The Handle, the Correlation i.d., and the Causality i.d. fields are largely responsible for enabling an Enter event to be linked with a Call event.

There are generally two kinds of events. There are asynchronous events, e.g. "this thing happened". And there are transition events, e.g. going from E0 to E1. When you have a transition event, you typically have a transition back. The user firing the event specifies a Correlation i.d., which enables the Call event to be identified with the Return event. The Call and Return have the same Correlation i.d., and the Enter and Leave have their own Correlation i.d. Each Correlation pair matches up exactly one pair of Enter/Leave and Call/Return to enable the VSA to understand how to match up the pairs.

Each event source has its own notion that correlates a CALL with a RETURN. For example, COM is able to generate a GUID based on the current execution context and processor. In an alternative embodiment, a Correlation i.d. could be generated using the time the CALL was made. Generation of a Correlation i.d. is typically simple but cannot really be generalized. Each IEC caller must pick its own scheme. Even within a currently implemented embodiment, several schemes for generating Correlation i.d.'s coexist.

Another key piece of information is the Causality i.d. This is normally provided by COM, but any entity can provide its own value if desired. Whenever a COM RPC is created, a GUID is created for that RPC. That information is tracked around the network, e.g. for purposes of identifying when a circular reference has been created. For the purposes of the present invention, it is used to match things up. It's basically a unique i.d. to identify a particular stream of calls and to sort them out. It says that this Call goes with this Return, and that this Enter goes with this Leave. The VSA knows from the Causality i.d. that these are all somehow interrelated.

In general, the Correlation i.d. operates on the events that are known to one machine, and the Causality i.d. operates on events that occur across machines.

A Handle is a way of referencing an individual instance of an entity. Handles are used by a calling entity to call (reference) a particular instance of an entity. Thus, the calling entity knows what Handle it is calling, and the entity being called (the target) knows its own Handle. When this process is applied for both the source and the target (each of which will have its own Handle), it is possible to collect together four events into the standard group of CALL/ENTER/LEAVE/RETURN. It is important to realize that any entity instance can have many different Handles that refer to it. For example, when A and C are both talking to B, A might use the Handle "BAT" to refer to B, where C might use the Handle "BALL" to refer to B.

From the information contained in the table shown in FIG. **6**, the VSA deduces that Call **170** goes with Return **176**, and that Enter **172** goes with Leave **174**. The VSA knows they're related. By knowing that the Source Handle **180** for Call **170** corresponds to Source Handle **186** for Enter **172**, and that Target Handle **182** for Call **170** corresponds to Target Handle **184** for Enter **172**, it knows that Call **170** is linked with Enter **172**. In similar fashion, the VSA determines that Enter **172** is linked with Leave **174**, and that Leave **174** is linked with Return **176**.

The table shown in FIG. **6** will now be described in detail to illustrate how a relationship can be deduced between a source entity and a target entity. The table of FIG. **6** shows

a standard four-event transition sequence. This sequence is not the only possible one but is merely one example.

In this example, the CALL event fires, and the system is given fall information about the source but only knows the target Handle is H1. When the target fires the ENTER event, two deductions can be made: (1) the CALL event can now be filled in, and (2) Handle H1 (the target) has now been defined to be M1, P1, E1, I1. So the CALL event is now completely specified. Additionally, the ENTER event uses Handle H0 which was previously defined to be **M0, P0, E0, I0**, and so the ENTER event can be completely filled in too.

When the LEAVE event arrives again from the target, two more deductions can be made: (1) the source information for the LEAVE event can be filled in by noticing that Handle H0 has previously been defined to mean **M0, P0, E0, I0**, and (2) we can now deduce that this LEAVE event and the previous ENTER event are a pair, because they have the same Correlation i.d. (i.e. "CB").

When the final RETURN event arrives, three deductions can be made: (1) we can fill in the target information for the RETURN event, because we know that H1 means **M1, P1, E1, I1**, (2) we can pair this RETURN up with the previous CALL by noticing that the Correlation i.d. ("CA") matches that of the CALL event, and (3) all four events are a set because their Causality i.d. is the same, and they have two pairs of matching Correlation i.d.'s.

The proper choice of a Handle depends in part on the entity causing the event. As in the case of a Correlation i.d., the generation of a Handle is typically simple but cannot really be generalized. Several routine schemes for generating Handles exist within a currently implemented embodiment of the invention.

It generally takes all three pieces of information together in context to create a functional diagram of how all of the pieces communicate. No single piece of information is vital to successful analysis. Dropping one or more fields still allows an implemented embodiment of the invention to generate useful analysis data. However, the removal of all source information makes it impossible to recognize a transition, for example, and thus impossible to diagram transitions in the system. Similarly, the loss of critical data such as the Correlation i.d. makes it impossible to draw a tree of events.

It will be understood by one of ordinary skill that other options for ensuring that a source and a target can appropriately identify themselves are possible.

Triggers

FIG. **7** illustrates in schematic fashion how events selected by a user are monitored. Triggers enable the VSA user to watch for a selected condition or error to occur. In many cases, a developer knows that an error will occur, but he or she doesn't know exactly when it will occur. The present invention allows the developer to set a trigger for collecting data in these situations.

Triggers can be set either for conditions for which an IEC creates an event, such as "a COM event in Machine A", or for conditions for which a DEC creates an event, such as PerfMon data reflecting CPU utilization.

The user can use Boolean operators, for example "OR" and "AND", to specify a set of two or more trigger conditions to watch. For example, a client can request to be alerted when a first designated CPU utilization OR a second designated CPU utilization exceeds 75%. Alternatively, an alert could happen when CPU utilization exceeded 75% AND disk utilization was less than 10%, potentially highlighting the need to obtain additional processing power.

A developer can also specify a first filter for "normal" event-monitoring, and a second filter (which can be more detailed or comprehensive than the first filter) to apply when the trigger condition occurs. A "filter" is a way in which the system user can specify what is to be monitored in the system under examination. Filters will be discussed in greater detail below in the sub-sections entitled "Filter Reduction", "Filter Combination", and "Filter Specification".

In **FIG. 7** an LEC **192** is depicted monitoring an application **190**. Events created by IECs and DECs (not illustrated in **FIG. 7**) are collected by LEC **192**. Upon the occurrence of a trigger condition, LEC **192** dumps the events to the VSA **100** or else signals an alert to the VSA **100**.

While watching for one or more trigger condition(s), event monitoring continues as usual, but data only requested by the trigger filter is not logged, while data requested by the monitoring filter continues to be logged as normal.

While waiting for a trigger condition to occur, events are retained transiently by the LEC **192** in a circular buffer whose size can be specified by VSA **100**. For example, VSA **100** can specify that the buffer store 500 events, so when the 501st event comes in, the first event is written over.

When the user's specified trigger condition is detected, the LEC **192** can immediately transmit all of the buffered events to the VSA **100** for logging. These provide data about the application prior to the failure or other condition. In addition, the LEC **192** can start collecting more events at a higher rate (in accordance with the second filter, for example) which events provide additional detailed information.

VSA **100** can also specify a reset condition, either as part of the second filter or as a separate filter. When the reset condition is met, the LEC **192** returns to the low-impact minimal collection condition specified by the first filter and once again monitors for a trigger condition.

It will be apparent to one of ordinary skill in the art that suitable data compression techniques can be applied to increase the efficiency of the event buffering and data transmission aspects of the invention. Data compression can be used both for storing events and for sending large quantities of events or event-related data through the data processing system.

Data Security

Information that is processed by a system performance analysis tool is likely to be confidential. Like any debugging tool, the VSA should ensure that the debuggability of the system cannot become a security hole. Additionally, VSA debugging is a shared resource in a distributed environment. As such, it is important that proper security precautions be taken to prevent malicious users from obtaining this data.

The invention provides a secure environment for data collection through the use of discretionary access controls. These access controls can be applied, at the discretion of the user, to the collection of data from a specific machine, to the monitoring of specific entities, and to the collection of specific events.

In one aspect of the invention VSA **100** is implemented as a DCOM server which can be configured to run as any identity, so it can control the resources and information it has access to. In addition, the server can run in a Windows NT authenticated domain, so that access to the server can be

controlled by discretionary access controls based on authentication identities.

It will be apparent to one of ordinary skill in the art that discretionary access enforcement can be based on the processes desired to be monitored effectively. It will also be apparent to one of ordinary skill in the art that suitable encryption techniques can be employed to enhance security within the VSA. Since DCOM is used to communicate with the server, standard RPC encryption can be used. In addition, the use of COM's custom marshalling allows for any virtually any type of encryption technology to be used.

Filter Reduction

FIG. **8** illustrates a process of filter reduction as used within an exemplary embodiment of the invention. First, the use of filters within the context of the invention will be discussed. VSA users specify the desired information to monitor via a User Filter **200**. That is, a filter defines what information the VSA will collect and analyze. Users can specify this information in a "system" scope, for example, "All COM and ADO events from Machines A and B". In addition to directing a filter to a machine, a filter can be directed to a process, component (e.g. ADO), EEC, DEC, event, thread, or to multiples or combinations of the foregoing.

The user filter **200** can comprise a filter **202** for Machine A, which in turn can comprise filters **204**, **206**, **208** for Processes A1, A2, A3, respectively. Likewise user filter **200** can comprise a filter **212** for Machine B that in turn comprises filter **214**, **216**, **218** for Processes **B1**, **B2**, **B3**, respectively.

A filter can generally be expressed as a single Boolean expression in a set of unbound variables. These variables communicate to the data provider with events, and to the event sources and their categories. Using the example above, the filter would be (Machine=A OR Machine=B) AND (EventSource=COM OR EventSource=ADO).

Filter reduction is a process employed by the VSA to extract portions of a filter relevant to specify a specific portion of the monitoring infrastructure. Using the previous example, the filter would be reduced by "Machine A" and then "Machine B" to determine the filter fragments that are specific to each machine. These fragments are transmitted to the LECs. The LECs, in turn, reduce the filter by the registered entities/processes on the system. The result is a filter fragment that can be used to determine if a specific data source is enabled or disabled. This information is communicated to the IECs to provide the efficient IsActive function.

Filter reduction is the process of modifying or creating a new version of a Boolean expression by binding a subset of the variables within the expression. For example, if the example filter above is sent to machine C, the Machine=A clause can be reduced to FALSE, and the Machine=B clause can be reduced to FALSE. Since the expression "FALSE AND anything" is FALSE, the whole expression evaluates to FALSE for machine C, meaning that all collection infrastructure on machine C can be deactivated.

Another example of filter reduction would be to reduce the example filter ("All COM and ADO events from Machines A and B") by "Machine=A". This results in the filter "EventSource=COM OR EventSource=ADO". Thus the result of this filter reduction is a Boolean expression, not just a TRUE or FALSE expression.

The LECs also make use of a specialized form of filter reduction to determine which dynamic data is desired. Collection and transmission of dynamic data is expensive, and a filter is scanned for clauses that specifically refer to the dynamic information that is required.

The VSA is communicating with multiple LECs, and to operate efficiently it reduces the filter from a global scale down to a filter for a particular machine. What goes into an LEC is that portion of the filter that pertains to a particular machine.

At the next level the LEC breaks the information into pieces which are germane to each IEC to identify whether or not that IEC should be turned on or off. So filter reduction occurs on at least two levels. The first level of filter reduction occurs at the VSA itself. The second level occurs at the LEC, which decides which IEC to turn on or off. It will be apparent to one of ordinary skill in the art that a third level could be at the IEC level.

If at any point in the reduction the VSA determines that the filter is guaranteed to be False for a given machine, the collection mechanism is turned off on that machine. If a filter specifying "Machine=A and Process=7" is sent to Machine B, it's just False. Data collection for Machine B is left off and not turned on, which lets Machine B operate more efficiently. On Machine A the collection mechanism is left off for everything except Process 7. This is similar to binding variables in a Boolean expression. If it's either True or False, you know what to do. But if it's undefined, you have to send the expression further down the chain. This feature applies to processes and components as well. It will be apparent to one of ordinary skill in the art that it could be applied to any level, from the machine level down to the thread level.

A machine-specific filter can be broadcast to a given machine. Generally, the reduction is performed at the client machine, and then the reduced filter is broadcast to specific machines. Again, it will be apparent to one of ordinary skill in the art that specific filters can be applied to any level.

A third level of filter reduction can occur in the DEC. The DEC can specify exactly what pieces of information are being looked for. For example, an event monitoring application such as PerfMon can collect about 7000 pieces of information, and it's very expensive to collect each one. So the filter needs to be reduced further by identifying exactly which pieces of information to collect. In the VSA user interface, the user can, if desired, be constrained to select PerfMon events a certain way, so they can't select them in complex Boolean expressions. When the filter makes its way through the network to the right creator, those PerfMon expressions are specifically referenced to the filter and collect exactly those expressions.

That combination of constraint in the VSA user interface and appropriate analysis of the results means that the VSA collects only those things specifically asked for in the dynamic case. This is important because every time a dynamic event is timed, one event can be fired every half second or every second, meaning a lot of events are fired. This can overwhelm the system infrastructure. So a filter reduction system is applied to the events that are initiated by the application. And extra reduction can be applied to events which are initiated by PerfMon. This could also be done for events at the IEC if desired.

Filter Combination

FIG. **9** illustrates a process of filter combination as used within an exemplary embodiment of the invention. It is possible, and likely, that multiple users will be monitoring applications running on shared servers. When this occurs, multiple filters can be issued to the same LEC. To ensure the most efficient collection, the LEC can combine all of the filters prior to performing the entity/process reduction.

With reference to FIG. **9**, a first user generates user filter **1** in box **231**, while a second user generates user filter **2** in box **232**. These filters are combined by the LEC into a merged or combined filter **235**, which in turn applies a filter for process A1 in box **236**, a filter for process A2 in box **237**, and a filter for process A3 in box **238**. The filters are reduced after they have been combined.

Appropriate IECs and DECs then monitor and collect events in accordance with the combined filter. One or more LECs, depending upon whether the items being monitored are on one or multiple machines, collect events from the IECs and DECs, in accordance with the combined filter, and send them to their respective requesting users, who may be on a single control station or at multiple control stations.

FIG. **10** illustrates another process of filter combination as used within an exemplary embodiment of the invention. With reference to FIG. **10**, filters for processes B1–B3 in boxes **246–248**, respectively, are combined in LEC **245** and passed on to users **1** and **2** in boxes **241** and **242**, respectively.

When events are collected by the LEC **245** from different sources within the data processing system under examination, it determines which clients are interested and routes the events to the respective clients who specified that the events be monitored. Because of the efficient and flexible nature of the filters, and the general-case nature of the reduction process described above, monitoring and collection from multiple machines imposes no extra performance overhead. Performance is simply as if all the monitoring were happening from a single machine.

Filter Specification

FIG. **11** illustrates a screen print of an exemplary user interface for specifying a filter. The VSA provides a large number of events that can be monitored. Consequently, an efficient mechanism is provided for the user to specify desired event data. The user interface (UI) of the invention provides a quick, easy graphical way for the user to specify the desired queries.

In the graphical UI, users are presented with three trees, each appearing in a separate window **250**, **252**, **254**, that represents the key information: a Machines/Processes window **250**, a Components window **252**, and a Categories/Events window **254**. The Machines/Processes window **250** presents all of the machines being monitored and the processes on the machines. The Components window **252** presents the registered VSA data sources on the machines being monitored. The Categories/Events window **254** identifies all of the registered VSA events that can be monitored. These can be organized hierarchically in a pre-defined structure, but the user can tailor it to his or her own structure and define his or her own events to be monitored.

It will be apparent to one of ordinary skill in the art that process threads could constitute another level of filter specification.

Event sources are required to pre-register which events they can emit when they are installed, and this information is transmitted at startup from the LEC to the central machine. By selecting the "Collect" tab **256**, the user can quickly select the desired information to analyze. More complex queries can be generated by creating groups of selections using the "OR" tab **258**. As the user makes selections, a textual representation of the query, appearing in text window **260**, is dynamically generated in synchronism with the graphical depiction in windows **250**, **252**, and **254**, so the user can verify his or her selection, and understand its behavior. Finally, the user can specify very sophisticated filter queries by entering the filter directly as text in text window **260**.

The tree-oriented part of the user interface allows highly complex filters to be created without a user having to understand the specific syntax or functionality. The system takes advantage of the fact that users have built-in understanding about the "rational" Boolean operators that are used to combine clauses ("OR" for bindings of the same variable, "AND" for bindings of independent variables). The same filter mechanism and user interface are used to both specify what to analyze and to refine the data which has been collected and which is presented to the user. VSA **100** analyzes data both as events are collected as well as after they have been collected. That is, users can filter already collected data, in a "post mortem" fashion, to create analysis reports of specific elements of the data without having to recollect the data.

The user can additionally specify debug and/or trace switches. These are run-time switches. They have a filter to determine the appropriate targets. Components, for example, can access the name/value pairs using the same interface as the IsActive and FireEvent status conditions.

Thus a user can chose which events to monitor. Boolean operators can be applied both within the windows and between the windows. Generally OR's are used within the windows, while AND's are used between the windows. In addition, the UI can enable the user to chose from a pre-defined list of the "top N" filters or queries, so that the user can quickly select from the top N.

Location of APIs

FIG. **12** illustrates a system level overview of an exemplary embodiment showing where APIs of the present invention can appear within the software architecture of a distributed computing system.

In a generalized and slightly over-simplified manner, the software architectures for two separate data processing system **301** and **302** are illustrated. Systems **301** and **302** each comprise a plurality of applications, represented by **310** and **340**, respectively. Systems **301** and **302** additionally each comprise software referred to as "middleware" identified by reference numbers **320** and **350**, respectively, and they each comprise operating system software **330** and **360**, respectively. The above-described software executes in the processor(s) of data processing systems **301** and **302**, the application programs running under the control of their corresponding operating systems.

It will be understood that applications **310**, **340**, middleware **320**, **350**, and the operating system software **330**, **360** can be entirely local to the data processing system **301** or **302**, or they can be distributed among data processing systems **301**, **302**, and additional data processing systems (not shown but implied by busses **322** and **342**).

Systems **301** and **302** can communicate with each other over bus **332**. Systems **301** and **302** can communicate with other systems (not shown) over busses **322** and **352**, respectively.

Each system **301** and **302** comprises APIs located in either the middleware or the operating system or in both. In a currently implemented embodiment, APIs are located in both. In order to facilitate utilization of the performance analysis tools of the present invention by software developers, APIs are provided to give a wide variety of functions, in the form of software modules and components, in common to a broad spectrum of applications. Any one

application typically uses only a small subset of the available APIs. Providing a wide variety of APIs frees application developers from having to write code that would have to be potentially duplicated in each application.

The APIs of the present invention offer the application developer ready access to the built-in performance analysis functions appearing in the middleware and operating system portions of the software architecture.

In the next section, various APIs are presented which allow applications to interface with various modules and components of the networking and operating system environment in order to implement the performance monitoring and analysis features of the invention.

Exemplary APIs and their Functions

This section presents and describes exemplary APIs relating to the performance monitoring and analysis features of the invention. It will be understood that these APIs are embodied on a computer-readable medium for execution on a computer in conjunction with an operating system or with middleware that interfaces with an application program having one or more event-generating components.

The APIs will first be described in functional terms. One or more applications, e.g. applications identified generally by reference number **310** or **340** in FIG. **12** are assumed to be running under the control of an operating system, e.g. operating system **330** or **360**. With respect to any one application program, in particular, the application can have any of a number of event-generating components. The application program utilizes APIs (such as APIs **325** or **355** located in middleware **320** or **350**, respectively, or APIs **335** or **365** located within operating systems **330** or **360**, respectively) associated with the event-generating component which operate to receive data from the operating system and to send data to the operating system.

This set of APIs includes a first interface that enables the operating system to set or disable a status condition ("IsActive") in the application, and it further includes a second interface that receives a status query from the operating system and that returns the status (True or False) of the status condition to the operating system.

The set of APIs includes an interface that enables the operating system to read any one or more of several fields in the application. These fields include arguments, causality i.d., correlation i.d., dynamic event data, exception, return value, security i.d., source component, source handle, source machine, source process, source process name, source session, source thread, target component, target handle, target machine, target process, target process name, target session, and target thread.

Now from the point of view of an operating system, consider that an operating system can have an event-registering or event-collecting component. The APIs also include an interface that enables the operating system to query whether a status condition ("IsActive") is set or disabled in the application, and they further include an interface that returns data to the operating system only if the status condition is set.

The APIs detailed below are described in terms of the C/C++ programming language. However, the invention is not so limited, and the APIs can be defined and implemented in any programming language, as those of ordinary skill in the art will recognize. Furthermore, the names given to the API functions and parameters are meant to be descriptive of their function. However, other names or identifiers could be associated with the functions and parameters, as will be apparent to those of ordinary skill in the art.

Four sets of APIs are presented: APIs for generating events (C interface), APIs for generating events (automation binding), APIs for registering events and sources (C binding), and APIs for registering events and sources (automation binding).

APIs for generating events used by applications that interface with the performance analysis functions of the present invention are presented below, both for C interface and for automation binding.

```
APIs for Generating Event (C Interface):
    HRESULT BeginSession(
        [in] REFGUID guidSourceID,
        [in] LPCOLESTR strSessionName
    );
        HRESULT EndSession(
        );
        HRESULT IsActive(
        );
        typedef [v1_enum] enum VSAParameterType {
            cVSAParameterKeyMask= 0x80000000,
            cVSAParameterKeyString=0x80000000,
            cVSAParameterValueMask=0x0007ffff,
            cVSAParameterValueTypeMask=0x00070000,
            cVSAParameterValueUnicodeString=0x00000,
            cVSAParameterValueANSIString=0x10000,
            cVSAParameterValueGUID=0x20000,
            cVSAParameterValueDWORD=0x30000,
            cVSAParameterValueBYTEArray=0x40000,
            cVSAParameterValueLengthMask=0xffff,
        } VSAParameterFlags;
        typedef [v1_enum] enum VSAStandardParameter {
                cVSAStandardParameterDefaultFirst=0,
                cVSAStandardParameterSourceMachine=0,
                cVSAStandardParameterSourceProcess=1,
                cVSAStandardParameterSourceThread=2,
                cVSAStandardParameterSourceComponent=3,
                cVSAStandardParameterSourceSession=4,
                cVSAStandardParameterTargetMachine=5,
```

-continued

```
        cVSAStandardParameterTargetProcess=6,
        cVSAStandardParameterTargetThread=7,
        cVSAStandardParameterTargetComponent=8,
        cVSAStandardParameterTargetSession=9,
        cVSAStandardParameterSecurityIdentify=10,
        cVSAStandardParameterCausalityID=11,
        cVSAStandardParameterSourceProcessName=12,
        cVSAStandardParameterTargetProcessName=13,
        cVSAStandardParameterDefaultLast=13,
        cVSAStandardParameterNoDefault=0x4000,
        cVSAStandardParameterSourceHandle=0x4000,
        cVSAStandardParameterTargetHandle=0x4001,
        cVSAStandardParameterArguments=0x4002,
        cVSAStandardParameterReturnValue=0x4003,
        cVSAStandardParameterException=0x4004,
        cVSAStandardParameterCorrelationID=0x4005,
        cVSAStandardParameterDynamicEventData=0x4006,
        cVSAStandardParameterNoDefaultLast=0x4006
} VSAStandardParameters;
typedef [vl__enum] enum eVSAEventFlags {
        cVSAEventStandard=0,
        cVSAEventDefaultSource=1,
        cVSAEventDefaultTarget=2,
        cVSAEventForceSend=8
} VSAEventFlags;
HRESULT FireEvent(
        [in] REFGUID guidEvent,
        [in] int nEntries,
        [in, size_is(nEntries)] LPDWORD rgKeys,
        [in, size_is(nEntries)] LPDWORD rgValues,
        [in, size_is(nEntries)] LPDWORD rgTypes,
        [in] DWORD dwTimeLow,
        [in] LONG dwTimeHigh,
        [in] VSAEventFlags dwFlags
);
}
```

"BeginSession" is called by an entity before it fires events to register its entity and instance names (source and session).

"EndSession" is called by an entity after it completes firing events.

"IsActive" is called by an entity which is considering firing events and wishes to know if anyone is listening.

"FireEvent" fires an actual event from an entity.

```
APIs for Generating Events (Automation Binding):
    HRESULT BeginSession(
        [in] BSTR guidSourceID,
        [in] BSTR strSessionName
    );
    HRESULT EndSession(
    );
    HRESULT IsActive(
        [out] VARIANT_BOOL *pblsActive
    );
    HRESULT FireEvent(
        [in] BSTR guidEvent,
        [in] VARIANT rgKeys,
        [in] VARIANT rgValues,
        [in] long rgCount,
        [in] VSAEventFlags dwFlags
    );
}
```

The comments for the above set of "APIs For Generating Events" are the same as for the C Interface APIs preceding them.

APIs for registering events and sources used by applications that interface with the performance analysis functions of the present invention are presented below, both for C interface and for automation binding.

```
APIs for Registering Events and Sources (C Interface):
    HRESULT RegisterSource(
        [in] LPCOLESTR strVisibleName,
        [in] REFGUID guidSourceID
    );
    HRESULT IsSourceRegistered(
        [in] REFGUID guidSourceID
    );
    HRESULT RegisterStockEvent(
        [in] REFGUID guidSourceID,
        [in] REFGUID guidEventID
    );
    HRESULT RegisterCustomEvent(
        [in] REFGUID guidSourceID,
        [in] REFGUID guidEventID,
        [in] LPCOLESTR strVisibleName,
        [in] LPCOLESTR strDescription,
        [in] long nEventType,
        [in] REFGUID guidCategory,
        [in] LPCOLESTR strIconFile,
                        [in] long nIcon
    );
    HRESULT RegisterEventCategory(
        [in] REFGUID guidSourceID,
        [in] REFGUID guidCategoryID,
        [in] REFGUID guidParentID,
        [in] LPCOLESTR strVisibleName,
        [in] LPCOLESTR strDescription,
        [in] LPCOLESTR strIconFile,
        [in] long nIcon
    );
    HRESULT UnRegisterSource(
        [in] REFGUID guidSourceID
    );
    HRESULT RegisterDynamicSource(
```

-continued

```
        [in] LPCOLESTR strVisibleName,
        [in] REFGUID guidSourceID,
        [in] LPCOLESTR strDescription,
        [in] REFGUID guidClsid,
        [in] long inproc);
    HRESULT UnRegisterDynamicSource(
        [in] REFGUID guidSourceID);
    HRESULT IsDynamicSourceRegistered(
        [in] REFGUID guidSourceID);
};
```

"RegisterSource" is called by code that is installing a new event-generating entity on a machine.

"IsSourceRegistered" detects if an event-generating entity is present.

"RegisterStockEvent" is called by an event-generating entity to note its use of a system event.

"RegisterCustomEvent" is called by an event-generating entity to note its definition of a custom event.

"RegisterEventCategory" is called by an event-generating entity to note its definition of a custom event category.

"UnRegisterSource" is called by code that is uninstalling an event-generating entity.

"RegisterDynamicSource" is called by code that is installing a DEC (dynamic event-generating entity).

"UnRegisterDynamicSource" is called by code that is uninstalling a DEC (dynamic event-generating entity).

"IsDynamicSourceRegistered" detects if an event-generating entity is present.

APIs for Registering Events and Sources (Automation Binding):

```
HRESULT RegisterSource(
    [in] BSTR strVisibleName,
    [in] BSTR guidSourceID
);
HRESULT IsSourceRegistered(
    [in] BSTR guidSourceID,
    [out] VARIANT_BOOL *pbIsRegistered
);
HRESULT RegisterStockEvent(
    [in] BSTR guidSourceID,
    [in] BSTR guidEventID
);
HRESULT RegisterCustomEvent(
    [in] BSTR guidSourceID,
    [in] BSTR guidEventID,
    [in] BSTR strVisibleName,
    [in] BSTR strDescription,
    [in] long nEventType,
    [in] BSTR guidCategory,
    [in] BSTR strIconFile,
    [in] long nIcon
);
HRESULT RegisterEventCategory(
    [in] BSTR guidSourceID,
    [in] BSTR guidCategoryID,
    [in] BSTR guidParentID,
    [in] BSTR strVisibleName,
    [in] BSTR strDescription,
    [in] BSTR strIconFile,
    [in] long nIcon
);
HRESULT UnRegisterSource(
    [in] BSTR guidSourceID
);
HRESULT RegisterDynamicSource(
    [in] BSTR strVisibleName,
    [in] BSTR guidSourceID,
```

-continued

APIs for Registering Events and Sources (Automation Binding):

```
        [in] BSTR strDescription,
        [in] BSTR guidClsid,
        [in] long inproc);
    HRESULT UnRegisterDynamicSource(
        [in] BSTR guidSourceID);
    HRESULT IsDynamicSourceRegistered(
        [in] BSTR guidSourceID,
        [out] VARIANT_BOOL *boolRegistered);
};
```

The comments for the above set of "APIs For Registering Events and Sources" are the same as for the C Interface APIs preceding them.

The APIs for registering events and sources (C interface/automation binding) can be used by an application to register which events can be generated by a data source. These APIs turn on and off such registration. They also specify whether the registration is a pre-defined, standard event or a custom event. They can also specify the event category, and they can determine whether a source is registered or not.

Automatic Generation of Animated Application Model

FIG. **13** illustrates a screen print of an animated application model which the present invention generates to show the structure and activity of an application whose performance is being studied. An important innovation in the VSA's analysis function is its ability to dynamically generate diagrams of the functionally active structure of the application.

The VSA creates the application diagrams by closely examining the event data that is received. As explained above, events are correlated by the VSA to understand the flow of control. The data design described above makes it possible to understand which events need to be correlated and how they should be grouped and connected.

Correlation makes use of the source and target information specified in the event data. When insufficient information is present, additional heuristics can be used to extrapolate the event flow. This includes time-ordering, COM causality information, and event handles.

With reference to the screen print **370** of FIG. **13**, the functional interrelationship among blocks such as blocks **371** and **372** is visually depicted. (It will be understood by one of ordinary skill in the art that, while all blocks in FIG. **13** are depicted with dummy labels, in practice each block will bear an appropriate label in accordance with that block's function or place within the performance model.) It will also be understood by one of ordinary skill that many other forms of visual portrayal of the application performance model can be used.

As new diagram elements are identified, they are added to the user's screen **370**. Frequently sufficient information is not available to immediately connect them to other entities on the diagram. This is the case with blocks **381** and **382** in FIG. **13**. As data becomes available, the entities are connected.

This application model diagram is highly interactive. Selections made in other VSA windows can result in selections in the diagram. Incoming events are directly animated into the diagram. Diagram blocks can be expanded or collapsed to show more or less detail.

To support this interactive behavior, the diagram data structures use a network of linked mapping tree data structures to efficiently understand the impact of new data, and to determine the blocks required to be added or removed when more data arrives.

Incomplete information is stored specially, and when other incomplete data arrives, there is an attempt to pair up the incomplete data using pre-defined heuristics and the data design described above.

Because the internal storage of the diagram only stores blocks and their connections, it is very space efficient. In normal scenarios storage space does not grow very fast proportionate to the number of events that have been viewed.

FIG. **14** illustrates various user interface features of an animated application model in an exemplary embodiment of the invention. The user interface features are shown generally by reference number **400**. In the UI depicted in FIG. **14**, diagrams are portrayed of the different blocks representing varying levels of detail of a hierarchical model of the application.

As shown in FIG. **14**, four different types of diagrams are available representing varying levels of detail: machines, processes, data sources, entities, and instances. Users can expand and collapse items on these diagrams to create the exact level of detail required. As well, the recorded event data can be depicted adjacent to the animated application model or overlaid upon it. In addition, using VCR-like commands, described below with reference to FIG. **14**, users can play and replay the application execution, stop, pause, reverse, speed up, slow down, and so forth.

Merely by way of illustration, an animated application model, shown generally by reference number **410**, includes a machine **404**, which is shown coupled functionally to a machine **412**, which in turn is coupled to a machine **411**. Each machine **404**, **411**, **412** can, in turn, be coupled to other items (not shown).

A visual depiction of a first machine **404** can be "exploded" into its constituent processes, depicted by box **402**. The user can further "drill" into a process, such as Process #1, to explode its constituent entities, depicted by box **406**. Further, the user can drill into an entity, such as Entity #1, for example, to explode a view, depicted by box **408**, showing the various Instances #1 through #N which are included in Entity #1.

The drill-in shown in FIG. **14** can be mixed in the same user screen. That is, a drill-in for machine **411** could show only its constituent processes, and a drill-in for machine **412** could show only its constituent processes plus the entities for one of the processes. So any individual box can be drilled down or up independently. In addition, the user can perform zooming, printing, and any other known screen operations.

The graphical UI includes a display and a user interface selection device, such as a keyboard or mouse. A model of the functionally active structure of the data processing system is displayed. Using the user interface selection device, a selection signal is generated with respect to a portion of the animated model, along with the user's expansion or contraction command. The VSA performs an expansion or contraction function on the selected portion in response to the selection signal and to the expansion or contraction command, and the selected portion is either exploded or contracted per the expansion or contraction command.

Behind this visual depiction of the application model, the VSA maintains a log of all of the events that have been collected.

The VSA utilizes a graphical UI paradigm in the form of a video cassette recorder (VCR) having, for example, Reverse, Stop, Pause, Speed, and Play commands. Other appropriate commands can be provided as indicated by an unlabeled button on the control panel. Using the VCR paradigm to control the depiction of the application performance, the VSA can run through each of the events and correspondingly animate the application model shown in FIG. **13** or FIG. **14**. For example, if the current event is between Machine #1 and Machine #N, then a connection segment **411** is highlighted. Using the VCR commands, the user can change the speed, pause the display, and go backward and forward.

While the user is doing this, a separate, adjacent window **430** shows the event details. So while the event is occurring, and the application model diagram of FIG. **14** is being animated, the user can also view other pertinent performance details in window **430**.

Also shown in FIG. **14** is an adjacent time line window **440** having equally spaced vertical lines throughout the time duration of an event. A special marker **445** moves from left to right through the vertical lines to show the progress of an event, either as the event occurs, or as the event is being played back by the user.

All of the windows are time-synchronized to one another.

Performance Analysis

FIG. **15** illustrates a representative display of performance data in an exemplary embodiment of the invention.

The VSA provides another important component for automatic analysis of collected data, the performance analysis component. The performance analysis component analyzes the collected data and creates a call tree by pairing events (e.g. Call and Return) and ordering them using temporal ordering and heuristics. The result is a presentation of the call tree in a Gantt style view with any Perfmon (or other dynamic) data displayed adjacent to or overlying the displayed call tree. With this view, the VSA provides a mechanism to simultaneously view application and environmental performance information and quickly drill into the details (by expanding to another level in the call tree). When the VSA is used to track and graph load information, the VSA provides an innovative way for the user to view how applications perform, behave, and degrade under different load and stress scenarios.

Like the animated application model, the call tree is generated by the application of suitable pre-determined heuristics, since the user does not have any a priori knowledge of the call relationships of more than two objects. Temporal and contextual information, for example, are used to deduce a call tree without full information. It will be apparent to one of ordinary skill that other kinds of information can also be used to deduce a call tree.

With reference to FIG. **15**, an upper window **450** includes a process summary portion **460** and a performance summary portion **470**. The process summary portion **460** comprises a Call Hierarchy including Call, Enter, Leave, and Return events. Each of these events can contain sublevels, as shown for the Call event. It will be understood that the sublevels can be further subdivided to whatever degree is required, as shown for the Leave event. The user can expand or collapse the levels of detail for each of the events, as desired.

Each of the Call, Enter, Leave, and Return events can have a corresponding Gantt type of representation, as illustrated in performance summary portion **470**, showing the duration of the event. For example, Gantt segment **471**

represents the duration of the Call event. The duration of the Enter, Leave, and Return events are shown by Gantt segments **472**, **473**, and **474**, respectively.

Performance summary portion **470** thus provides a GANTT-style presentation of the call tree, i.e. who calls whom. The GANTT bars **471**–**474** show when it started and how long the Call lasted. This information comes from the EEC.

Beneath the call tree performance summary, a graph **480** can be depicted to show, for example, the CPU utilization during the Call operation such as an RPC. Graph **480**, which may be positioned adjacent to or overlaying the Gantt segments **471**–**474**, could also illustrate any one or more other desired aspects of the system performance besides the CPU utilization. The Gantt chart can be based upon the application events. The graph can be selected from the time base.

Also shown in FIG. **15** is a summary window **490** which provides a distillation of what is shown in the performance windows **410** and **430** of FIG. **14** and in the upper window **450** of FIG. **15**. For example, if the time slice between dashed lines **481** and **482** is selected for scrutiny, a summary performance graph **492** is generated for the selected time segment. Summary window **490** also contains a textual description of the application's performance during the specified time segment.

Thus the user can view a tightly synchronized, easily comprehensible graphical and textual analysis and representation of the application performance, in the form of the animated block diagram **410**, the Event Detail window **430**, and the Time Line window **440** of FIG. **14**, as well as the process summary portion **460** and the performance summary portion **470** of FIG. **15**. The summary window **490** ties everything together. Again, everything is time-synchronized.

In addition, all of the above windows can be operated to display the application performance in real time as well as "post mortem". This applies as well to the animated application models, as shown in the screen print of FIG. **13** and in window **410** of FIG. **14**, so that in real time as an application is being analyzed, one block will appear, then another, and then the interconnection between the two blocks. Blocks are dynamically added, removed, and moved, and the interconnections between them are dynamically changed to reflect changing conditions in the execution of the application. The diagram is kept up to date with what is really happening.

FIG. **16** illustrates a screen print **500** of an exemplary display of performance data. Screen print **500** depicts the percentage of CPU utilization for a selected group of processors. Window **504** shows a graph line **505** which, for example, depicts the percentage of CPU utilization (right-hand side) versus time (bottom side). In general, graph lines represent overlaid DEC data.

Window **502** depicts a list of events relating to the operation of the processors under scrutiny.

Window **506** depicts a legend or key to the information shown in window **504**. Window **506** indicates the source machines (all) as well as summary performance information (a minimum of 13 processors, a maximum of 100 processors, and an average of 49 processors executing simultaneously; currently 35 processors concurrently executing). Window **506** also comprises a "legend" **507** which provides a color key **508** to assist the user in identifying graph lines in window **504**, such as Gantt bars **510**, **511**, and **512**, or graph line **505**. While window **504** only shows one graph line **505**, more can be shown. Window **506** provides an indication of the source machines, maximum, minimum, average, and current value for each graph line shown in window **504**.

Additional Tools

The VSA provides a few other tools which, when used in conjunction with the features described above, provide additional insight into application performance.

FIG. **17** illustrates a screen print **520** of a timeline display of performance data. The timeline window presents a visual representation of the timing of all related events. Dark clumps **522** represent tight groupings of events, while spaces **524** represent possible under utilization of resources. Timeline **520** can be annotated to present event activity per machine or per process (or other system resource) using different colors. This allows users to visually identify both potential system-wide and per-machine bottlenecks. As playback or monitoring continues, the timeline **520** acts as a real-time indicator of the current system context.

FIG. **18** illustrates a screen print **530** of summary display of performance data. Similar to previously described summary window **490** in FIG. **15**, but depicting different information, the summary information in screen print **530** presents a distillation of all events selected by the VSA user. That is, if multiple events are selected, the unique elements (e.g. source and target machines, processes, entities, etc.) are displayed. This is very useful when a time range is selected either in the timeline or performance viewer. The summary window allows the user to see a quick tally of what is going on in the application. This is a particularly important view because of the large volumes of data generated while monitoring a system.

Synchronization

FIG. **19** illustrates a screen print **550** of several synchronized sets of performance data. Screen **550** comprises several windows, including an animated application model or process diagram **552**, an event log window **554**, CPU performance view window **556**, event viewing window **558**, a summary window **560**, and a time line window **562**.

The VSA ensures that all information presented to the user is cross-correlated. This provides instant synchronization. When the user selects an item (or set of items) in one window, all other windows can (based on user preference) automatically highlight the selection. This includes the selection of specific events, selection of all events in a specified time range, or selection of all events associated with a specified entity. However, if the user desires, auto-synchronization can be turned off for any one or more windows.

FIG. **19** illustrates this concept. Here, for example, the user made a time selection in the performance view window **556** (representing PerfMon data) over a period of time where CPU behavior was in question. The animated application model or process diagram **552** highlights the entities/processes involved in the selection. The event log window **554** highlights all events in the specified time range, part of which represent a call tree. The event viewing window **558** presents data on a single event (for multi-event selections it highlights the first event). The timeline window **562** highlights the specified time range as well as shows performance peaks, and the summary window **560** tallies the events in the time range and presents a summary.

Thus, while displaying the animated functional model **552**, the control station can also simultaneously display

items such as summary data **560**, time data **562**, event details **558**, and/or an event log or call tree **554**.

Window synchronization avoids a common problem with systems based on multiple windows. In a typical multi-window system, the user wants to have one or two windows fully visible, while others are invisible. Typically no context flows to or from invisible elements, despite the fact that the user may want this to happen. The VSA avoids this problem by creating a user notion of a shared selection (the 'AutoSelection'), and allows the user to subscribe windows to that selection. As a result, the user is not confused by the flow of context, and instead they find it predictable and natural.

The system level overview of the operation of an exemplary embodiment of the invention has been described in the Detailed Description. As described, the method and apparatus for analyzing the performance of a data processing system and, in particular, to an application running on a distributed data processing system, enable users to quickly and easily observe the operational performance of such a system without significantly impacting such performance.

Methods of Exemplary Embodiments of the Invention

The previous sections have described the structure and operation of various exemplary embodiments of the invention. In this section, the particular methods performed by such exemplary embodiments are described by reference to a series of flowcharts. These methods constitute computer programs made up of computer-executable instructions. Describing the methods by reference to flowcharts enables one skilled in the art to develop such programs including such instructions to carry out the methods on suitable computing systems (the processor of the computing systems executing the instructions from computer-readable media).

FIGS. **19–27** are flowcharts of methods to be performed according to exemplary embodiments of the invention. It will be understood by one of ordinary skill that the steps depicted in these flowcharts need not necessarily be performed in the order shown. It will also be understood that while the flowcharts have "Start" and "End" blocks, in general the processes they depict are continuously performed.

FIG. **20** A–C is a flowchart illustrating, in steps **601** through **612**, overall data collection architecture and how data is collected via the IECs, DECs, and LECs. The process begins with block **601**. In block **602** the operating system or middleware creates an IEC reference. In the next block **603**, the control station **100** creates an LEC.

Block **604** depicts that the LEC converts the IEC reference to an IEC. In block **605** the LEC is indicated as being capable, for example, of turning the IEC on or off by enabling or disabling its IsActive status condition.

In block **606** the control station **100** can turn a DEC on or off.

In block **607** an IEC collects events generated by a data source within the data processing system under scrutiny. The term "collect" herein broadly includes the IEC's function of creating events in response to certain conditions occurring within the process space it is monitoring.

In block **608** the LEC collects events from the IEC and sends them to the control station **100**.

In block **609** the DEC collects events that are generated on a time basis. The term "collect" herein broadly includes the DEC's function of creating events in response to monitoring certain time-valued system functions.

In block **610** the LEC collects data from the DEC and sends it to the control station **100**. Block **611** indicates that the LEC buffers a predetermined quantity of data and only stores the data on request of the control station **100**. The process ends in block **612**.

FIG. **21** A–B is a flowchart illustrating, in steps **615** through **625**, an exemplary embodiment of overall data design and how the VSA determines and maps relationships between entities. The process starts with block **615**. Next in block **616** events are identified by one or more pre-defined event fields and/or custom event fields. In block **617** events that are generated as a result of interactions among entities in the data processing system under scrutiny are collected. In block **618** an IEC monitors events and sends them to an LEC. In block **619** a DEC monitors time-based events and sends them to an LEC. In block **620** an LEC collects events and sends them to the control station. Next in block **621** the VSA analyzes the events and their event fields, and in block **622** the VSA determines the relationships among the entities, as described earlier. In block **623** the VSA maps the relationship among the entities, based in part on the content of the event fields. In block **624** the VSA generates a functional block diagram of the relationship among entities, and the process ends in block **625**.

FIG. **22** A–B is a flowchart illustrating, in steps **630** through **639**, an exemplary embodiment of triggers. The method starts in block **630**. In block **631** a control station specifies one or more trigger conditions, and it can specify, if desired, a Boolean relationship between two or more trigger conditions. The control station can also specify filters, for example a first filter and a second filter. The second filter can be more detailed and comprehensive than the first filter. The control station can also specify a reset condition. It can also specify how many events the LEC should store in its circular buffer store.

In block **632** an LEC collects events in accordance with the first filter while watching for a trigger condition, and in block **633** the LEC's buffer store stores events collected by the LEC. In block **634**, when the LEC detects a trigger condition, it sends the stored events to the control station, and in block **635** the LEC begins collecting events in accordance with the second filter and sending them to the control station. In block **636** the LEC watches for a reset condition. In block **637**, if the LEC detects a reset condition, it stops sending events to the control station, and in block **638** the LEC reverts to collecting events in accordance with the first filter and watching for another trigger condition. The process ends in block **639**.

FIG. **23** A—B is a flowchart illustrating, in steps **645** through **653**, an exemplary embodiment of filter reduction. The process begins in block **645**. In blocks **646–648**, a user specifies a filter, which process can take the form of a series of iterations of blocks **646–648**. In block **646** a menu or graphical user interface is displayed which lists one or more items representing machines, components, IECs, DECs, processes, events, and threads within the data processing system under examination. The user can chose a filter in the form of a Boolean expression comprising two or more items. In block **647**, the user selects his or her choice by generating a suitable menu entry selection signal using, for example, a mouse or keyboard. Block **648** indicates that step **647** is repeated, as necessary, until all desired filter items have been selected by the user.

Next in block **649** the filter is either sent to one or more specific machines, processes, IECs, DECs, events, or threads, or it is broadcast generally throughout the data

processing system. In block **650** the filter is applied to one or more specific machines, processes, IEC, DECs, events, and/or threads, in accordance with its user-selected variables. In block **651** an EEC and a DEC collect events in accordance with the filter. In block **652** the LEC collects events from the IEC and the DEC in accordance with the filter, and the LEC sends the collected events to a control station. The process ends in block **653**.

FIG. **24** A–B is a flowchart illustrating, in steps **660** through **668**, an exemplary embodiment of filter combination. The process begins in block **660**. In block **661**, one or more control stations specify more than one filter. Each filter designates one or more machines, processes, IECs, DECs, events, and/or threads. In block **662** the filters are sent to one or more LECs, each of which combines the filters it receives into a respective combined filter. Each combined filter applies to specific machines, processes, IECs, DECs, events, and/or threads. In block **663** an IEC collects events generated by a first data source within the data processing system under examination. In block **664** a DEC collects events that are generated on a time basis by a second data source within the data processing system under examination. In block **665** the IEC and DEC each collect events in accordance with a combined filter.

In block **666** the LEC collects events from the IEC and from the DEC in accordance with a combined filter, and the LEC sends the events to the control station or control stations which specified that the events be monitored. In block **667** the control station analyzes the events. The process ends in block **668**.

FIG. **25** A–B is a flowchart illustrating, in steps **670** through **680**, an exemplary embodiment of a user interface for specifying one or more filters. The process begins in block **670**. In block **671** a control station provides a graphical user interface (UI) to a user for enabling the user to specify at least one filter. In block **672** a menu is displayed listing items representing event-generating machines, event-generating components, and/or categories of events with the data processing system under examination.

In block **673** the VSA receives a menu entry selection signal indicative of a user interface selection device selecting one of the items to monitor. Block **674** indicates that step **673** is repeated, as necessary, until all desired items have been selected.

Block **675** indicates an alternate step to step **672**, in that the UI displays a pre-defined list of filters from which a user can specify at least one filter. The pre-defined list can be a "top 10" of the most popular filters in use, and it can be updated automatically by the VSA. Here the user has only to click on one filter, and it automatically includes a set of the items displayed in block **672**.

In block **676** a textual representation of the user-selected filter is displayed in a window. In addition, a window is provided in which the user can enter the filter directly in text format. In block **677** an IEC and a DEC each collect events in accordance with the user-selected filter. In block **678** an LEC collects events from the IEC and from the DEC, in accordance with the filter, and the LEC sends the events to the control station. In block **679** the control station either analyzes events collected by the LEC as the events are collected, or the LEC analyzes the events after the events have been collected (in post mortem fashion). The process ends in block **680**.

FIG. **26** A–C is a flowchart illustrating, in steps **690** through **700**, an exemplary embodiment of automatic generation of an animated application model. The process begins in block **690**. In block **691** an IEC collects events generated by a first data source within a data processing system under examination. In block **692** a DEC collects events that are generated on a time basis by a second data source within the data processing system under examination.

In block **693** an LEC collects events from the IEC and from the DEC and sends them to the control station. In block **694** the control station analyzes the events and displays a model of the functionally active structure of the data processing system under examination. While displaying the animated functional model, the control station can also simultaneously display items such as summary data, time data, event details, and/or a call tree. In block **695** the control station keeps updating the animated model in real time as it receives and analyzes events.

In block **696** the control station presents a user interface (UI) to the user in the form of a display, a user interface selection device, and uses a video cassette recorder (VCR) paradigm to enable the user to analyze the performance of the data processing system. The UI displays user-selectable commands, such as Play, Replay, Stop, Reverse, Pause, and Change Speed of the animated model. In block **697** the UI also enables the user to select one or more portions of the model and to either explode or enlarge a selected portion of the model to show more detail, or to contract or shrink a selected portion of the model to show less detail.

In block **699** the control station displays the active portions of the animated model in a visually distinctive manner, for example by highlighting them. The process ends with block **700**.

FIG. **27** A–C is a flowchart illustrating, in steps **710** through **720**, an exemplary embodiment of a user interface for displaying the performance analysis of the system under examination. The process begins in step **710**. In block **711** the control station analyzes events, for example events received from an LEC. In block **712** the control station displays a call tree of the functionally active structure of the data processing system under examination. In block **713** the control station can, while continuing to display the call tree, display time-synchronized items such as Gantt type charts, process summary data, performance summary data, and/or time data. In block **714** the control station updates the call tree in real time while it continues to receive events and analyze them.

In block **715** the user interface enables the user to select one or more portions of the call tree to analyze more closely. In blocks **716** and **717**, the UI enables the user to either explode or enlarge a selected portion of the model to show more detail, or to contract or shrink a selected portion of the model to show less detail. In block **718** the control station uses heuristics such as time-ordering, causality information, and event handles to generate and display the call tree. In block **719** the control station displays active portions of the animated model in a visually distinctive manner, for example by highlighting them, displaying them in a different color, or "flashing" them. The process ends in block **720**.

The particular methods performed by the significant exemplary embodiments of the invention have now been described with reference to the flowcharts of FIGS. **19–26**.

Conclusion

A method and apparatus for analyzing the performance of a data processing system have been described which overcome many of the disadvantages of prior known systems. The VSA collects application performance data by use of instrumentation within the application environment and

using an efficient, distributed collection architecture. By instrumenting the core application platform, the VSA can obtain information about the application without having to make changes to it.

The VSA enables the user to view an animated model of the application as it is running, as a set of interconnected black boxes. It does so without re-architecting or recompiling the original code.

The VSA includes an efficient mechanism for collecting and transmitting the data to a central log, and for streaming it to disk. A user interface is provided for detailed and specific selection of what to analyze, and the system is automatically configured to minimize impact based on the selection criteria. This information is distributed across the monitored systems and is used to efficiently collect analysis data.

In addition, the user is provided with automatic analysis tools to filter and view the operation of the application and to locate performance issues. A user display provides overlay and time-synchronized system performance data in any of a wide variety of user-specified formats. The VSA can be used for both live and post-mortem analysis.

As a consequence, this invention provides software developers, including developers of distributed component-based systems, with the ability to understand and analyze the behavior of their software while it is executing. The VSA can help find performance bottlenecks, understand system structure, and isolate behavioral problems.

Although specific embodiments have been illustrated and described herein, it will be appreciated by those of ordinary skill in the art that any arrangement which is calculated to achieve the same purpose may be substituted for the specific embodiments shown. This application is intended to cover any adaptations or variations of the present invention.

It will be apparent to those of ordinary skill that the collection aspects of the invention can be implemented either in the operating system or in middleware. Furthermore, the implementation can be implemented in any desirable manner, e.g. by splitting it into separate pieces such as filter-specifying, event-firing, data collection, and analysis/presentation. For example, by including one or more pieces in the operating system, the potential utilization of the invention can be widespread.

For example, those of ordinary skill within the art will appreciate that in one embodiment a virtual-machine style system (e.g. a Java system) could automatically insert the implementing features of this invention into all programs at the virtual machine level.

Alternatively, a hardware-based system could automatically generate out-of-band signals at the hardware level in accordance with the concepts disclosed herein.

In addition, a data-bound system (e.g. an Oracle database) could use data triggers to get similar results.

Finally, as future operating systems are developed, the innovations herein could be applied to an agent-based operating system that is able to automatically migrate to different machines.

Therefore, it is manifestly intended that this invention be limited only by the following claims and equivalents thereof.

APPENDIX I

ADDITIONAL DESCRIPTION

The following material is not to be construed as pending claims

Data Design

1. A system for analyzing and mapping relationships among entities in a data processing system in which events are generated as a result of interactions among the entities comprising:

an event concentrator that collects the events; and

a control station coupled to the event concentrator and receiving events therefrom, the control station analyzing the events and mapping relationships among the entities.

2. A system as recited in claim 1 and further comprising:

an in-process event creator that monitors events and sends them to the event concentrator.

3. A system as recited in claim 1 and further comprising:

a dynamic event creator that monitors time-based events and sends them to the event concentrator.

4. A system as recited in claim 1 in which an event is identified by one or more event fields, and in which the control station analyzes events and maps relationships among entities based in part on the content of the event fields.

5. A system as recited in claim 3 in which an event field is from a group comprising arguments, unique i.d., causality i.d., correlation i.d., dynamic event data, exception, return value, security i.d., source component, source handle, source machine, source process, source process name, source session, source thread, target component, target handle, target machine, target process, target process name, target session, and target thread.

6. A system as recited in claim 5 in which an event field comprises one or more default event fields.

7. A system as recited in claim 4 in which an event field is a custom field.

8. A system as recited in claim 4 in which the events comprise a call event and an enter event between two entities, one being a source entity that performs the call event and the other being a target entity that performs the enter event, and in which the event fields comprise a unique i.d., causality i.d., and correlation i.d.

9. A system as recited in claim 8 in which the event fields further comprise source component, source machine, source process, and source session for the source entity.

10. A system as recited in claim 8 in which the event fields further comprise target component, target machine, target process, and target session for the target entity.

11. A system as recited in claim 1 in which the control station maps the relationship among the entities in the form of a functional block diagram.

12. A system as recited in claim 1 in which the data processing system is a distributed system comprising a plurality of machines, and entities reside on different machines.

13. In a data processing system comprising a plurality of entities, a method comprising the steps of:

collecting events that are generated as a result of interactions among the entities;

analyzing the events; and

determining the relationship among the entities.

14. The method recited in claim 13 and further including the step of:

mapping the relationship among the entities.

15. The method recited in claim **14** wherein the step of mapping the relationship among the entities comprises generating a functional block diagram of such relationship.

16. The method recited in claim **13**, wherein the steps recited therein can be performed in any suitable order.

17. A computer-readable medium having computer-executable instructions for performing the steps recited in claim **13**.

18. A method for analyzing and mapping relationships among entities in a data processing system in which events are generated as a result of interactions among the entities, and in which events are identified by one or more events fields, the method comprising the steps of:

collecting the events;

analyzing the events fields; and

mapping relationships among the entities based in part on the content of the event fields.

19. The method recited in claim **18**, wherein the steps recited therein can be performed in any suitable order.

20. A computer-readable medium having computer-executable instructions for performing the steps recited in claim **18**.

21. In a data processing system comprising a plurality of entities, an event concentrator, and a control station coupled to the event concentrator, the method comprising the steps of:

the event concentrator collecting events that are generated as a result of interactions among the entities; and

the control station analyzing the collected events and determining the relationship among the entities.

22. The method recited in claim **21** and further including the step of:

the control station mapping the relationship among the entities.

23. The method recited in claim **22** wherein the step of mapping the relationship among the entities comprises generating a functional block diagram of such relationship.

24. The method recited in claim **21** in which an event is identified by one or more event fields, and in which the control station analyzes events and maps relationships among entities based in part on the content of the event fields.

25. The method recited in claim **24** in which an event field is from a group comprising arguments, unique i.d., causality i.d., correlation i.d., dynamic event data, exception, return value, security i.d., source component, source handle, source machine, source process, source process name, source session, source thread, target component, target handle, target machine, target process, target process name, target session, and target thread.

26. The method recited in claim **24** in which an event field comprises one or more default event fields.

27. The method recited in claim **24** in which an event field is a custom field.

28. The method recited in claim **24** in which the events comprise a call event and an enter event between two entities, one being a source entity that performs the call event and the other being a target entity that performs the enter event, and in which the event fields comprise a unique i.d., causality i.d., and correlation i.d.

29. The method as recited in claim **28** in which the event fields further comprise source component, source machine, source process, and source session for the source entity.

30. The method as recited in claim **28** in which the event fields further comprise target component, target machine, target process, and target session for the target entity.

31. The method recited in claim **21** wherein the data processing system further comprises an in-process event creator, the method further comprising the step of:

the in-process event creator monitoring events and sending them to the event concentrator.

32. The method recited in claim **21** wherein the data processing system further comprises a dynamic event creator, the method further comprising the step of:

the dynamic event creator monitoring time-based events and sending them to the event concentrator.

33. The method recited in claim **21** in which the data processing system is a distributed system comprising a plurality of machines, and entities reside on different machines.

34. The method recited in claim **21**, wherein the steps recited therein can be performed in any suitable order.

35. A computer-readable medium having computer-executable instructions for performing the steps recited in claim **21**.

Triggers

1. A system for analyzing the performance of a data processing system that produces events, the system comprising:

a control station that analyzes events, the control station specifying at least one trigger condition;

an event concentrator, coupled to the control station, that collects events and watches for the at least one trigger condition; and

a store, coupled to the event concentrator, that stores events until the occurrence of the at least one trigger condition, whereupon the stored events are sent to the control station.

2. The system recited in claim **1**, wherein the store is a circular buffer, and the control station specifies how many events to store in the buffer.

3. The system recited in claim **1**, wherein the control station specifies two trigger conditions and a Boolean relation between them.

4. The system recited in claim **1**, wherein when the at least one trigger condition occurs, the event concentrator begins collecting events at a higher rate.

5. The system recited in claim **1**, wherein the control station specifies a reset condition, and wherein the event concentrator watches for a reset condition.

6. The system recited in claim **5**, wherein events are no longer sent to the control station when the event concentrator detects a reset condition

7. The system recited in claim **6**, wherein, upon occurrence of a reset condition, the event concentrator reverts to collecting events and watching for the at least one trigger condition.

8. The system recited in claim **1**, wherein the store utilizes data compression for storing the events.

9. The system recited in claim **1**, wherein the system utilizes data compression for sending the events.

10. A system for analyzing the performance of a data processing system that produces events, the system comprising:

a control station that analyzes events, the control station specifying at least one trigger condition, and the control station further specifying at least one filter;

an event concentrator, coupled to the control station, that collects events in accordance with the at least one filter and that watches for the at least one trigger condition; and

a store, coupled to the event concentrator, that stores events until the occurrence of the at least one trigger condition, whereupon the stored events are sent to the control station.

11. The system recited in claim **10**, wherein the store is a circular buffer, and the control station specifies how many events to store in the buffer.
12. The system recited in claim **10**, wherein the control station specifies two trigger conditions and a Boolean relation between them.
13. The method recited in claim **10**, wherein when the at least one trigger condition occurs, the event concentrator begins collecting events at a higher rate.
14. The system recited in claim **10**, wherein the control station specifies a reset condition, and wherein the event concentrator watches for a reset condition.
15. The system recited in claim **14**, wherein events are no longer sent to the control station when the event concentrator detects a reset condition.
16. The system recited in claim **15**, wherein, upon occurrence of a reset condition, the event concentrator reverts to collecting events and watching for the at least one trigger condition.
17. The system recited in claim **10**, wherein the control station specifies a first filter and a second filter, and the event concentrator collects events in accordance with the first filter until the at least one trigger condition occurs, whereupon the event concentrator collects events in accordance with the second filter.
18. The system recited in claim **17**, wherein the control station specifies a reset condition, and wherein the event concentrator watches for a reset condition.
19. The system recited in claim **18**, wherein events are no longer sent to the control station when the event concentrator detects a reset condition.
20. The system recited in claim **18**, wherein, upon occurrence of a reset condition, the event concentrator reverts to collecting events in accordance with the first filter and watching for the at least one trigger condition.
21. A method for analyzing the performance of a data processing system that produces events and that comprises a control station that analyzes events, an event concentrator, and a store, the method comprising the steps of:

the control station specifying at least one trigger condition;

the event concentrator collecting events while watching for the at least one trigger condition;

the store storing events collected by the event concentrator; and

the event concentrator sending the stored events to the control station upon the occurrence of the at least one trigger condition.

22. The method recited in claim **21**, wherein the store is a circular buffer, and the control station specifies how many events to store in the buffer.
23. The method recited in claim **21**, wherein the control station specifies two trigger conditions and a Boolean relationship between them.
24. The method recited in claim **21**, wherein when the at least one trigger condition occurs, the event concentrator begins collecting events at a higher rate.
25. The method recited in claim **21**, wherein when the at least one trigger condition occurs, the event concentrator begins sending events to the control station.
26. The method recited in claim **21**, wherein the control station specifies a reset condition, and wherein the event concentrator watches for a reset condition.
27. The method recited in claim **26**, wherein the event concentrator no longer sends events to the control station when the event concentrator detects a reset condition.
28. The method recited in claim **26**, wherein, upon occurrence of a reset condition, the event concentrator reverts to collecting events and watching for the at least one trigger condition.
29. The method recited in claim **21**, wherein the store utilizes data compression for storing the events.
30. The method recited in claim **21**, wherein the event concentrator utilizes data compression for sending the events.
31. The method recited in claim **21**, wherein the steps recited therein can be performed in any suitable order.
32. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 21.
33. A method for analyzing the performance of a data processing system that produces events and that comprises a control station that analyzes events, an event concentrator, and a store, the method comprising the steps of:

the control station specifying at least one trigger condition and at least one filter;

the event concentrator collecting events in accordance with the at least one filter while watching for the at least one trigger condition;

the store storing events collected by the event concentrator; and

the event concentrator sending the stored events to the control station upon the occurrence of the at least one trigger condition.

34. The method recited in claim 33, wherein the store is a circular buffer, and the control station specifies how many events to store in the buffer.
35. The method recited in claim 33, wherein the control station specifies two trigger conditions and a Boolean relationship between them.
36. The method recited in claim 33, wherein when the at least one trigger condition occurs, the event concentrator begins collecting events at a higher rate.
37. The method recited in claim 33, wherein when the at least one trigger condition occurs, the event concentrator begins sending events to the control station.
38. The method recited in claim 33, wherein the control station specifies a reset condition, and wherein the event concentrator watches for a reset condition.
39. The method recited in claim 38, wherein the event concentrator no longer sends events to the control station when the event concentrator detects a reset condition.
40. The method recited in claim 38, wherein, upon occurrence of a reset condition, the event concentrator reverts to collecting events and watching for the at least one trigger condition.
41. The method recited in claim 33, wherein the control station specifies a first filter and a second filter, and further comprising the steps of:

the event concentrator collecting events in accordance with the first filter until the at least one trigger condition occurs;

then the event concentrator collecting events in accordance with the second filter.

42. The method recited in claim 41, wherein the control station specifies a reset condition, and wherein the event concentrator watches for the reset condition.
43. The method recited in claim 42, wherein the event concentrator no longer sends events to the control station when the event concentrator detects a reset condition.

44. The method recited in claim 42, and further comprising the step of:

upon occurrence of a reset condition, the event concentrator reverts to collecting events in accordance with the first filter and watching for the at least one trigger condition.

45. The method recited in claim 33, wherein the steps recited therein can be performed in any suitable order.

46. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 33.

Filter Reduction

1. A system for analyzing the performance of a data processing system that produces events, the system comprising:

a control station that analyzes events, the control station specifying at least one filter; and

an event concentrator, coupled to the control station, that collects events in accordance with the filter.

2. The system recited in claim **1**, wherein the at least one filter is a Boolean expression comprising two or more events.

3. The system recited in claim **2**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

4. The system recited in claim **1**, wherein the data processing system further comprises a plurality of machines, and wherein the at least one filter is directed to at least one of the machines.

5. The system recited in claim **4**, wherein the at least one filter is a Boolean expression comprising two or more machines.

6. The system recited in claim **5**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

7. The system recited in claim **1**, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and wherein the at least one filter is directed to at least one of the processes.

8. The system recited in claim **7**, wherein the at least one filter is a Boolean expression comprising two or more processes.

9. The system recited in claim **8**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

10. The system recited in claim **1**, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and wherein the at least one filter is directed to at least one of the machines and at least one of the processes.

11. The system recited in claim **10**, wherein the at least one filter is a Boolean expression comprising two or more processes.

12. The system recited in claim **11**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

13. The system recited in claim **1**, wherein the at least one filter designates one or more machines, processes, in-process event creators, dynamic event creators, events, or threads.

14. The system recited in claim **1**, wherein the at least one filter is a Boolean expression.

15. The system recited in claim **14**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

16. The system recited in claim **1**, wherein the at least one filter is sent to one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

17. The system recited in claim **1**, wherein the at least one filter is broadcast throughout the data processing system, wherein it is applied by one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

18. A system for analyzing the performance of a data processing system that produces events, the system comprising:

a control station that analyzes events, the control station specifying at least one filter;

an in-process event creator that collects events generated by a first data source within the data processing system;

a dynamic event creator that collects events that are generated on a time basis by a second data source within the data processing system; and

an event concentrator collecting events from the in-process event creator and from the dynamic event creator, in accordance with the at least one filter, and sending the events to the control station.

19. The system recited in claim **18**, wherein the at least one filter is a Boolean expression comprising two or more events.

20. The system recited in claim **19**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

21. The system recited in claim **18**, wherein the data processing system further comprises a plurality of machines, and wherein the at least one filter is directed to at least one of the machines.

22. The system recited in claim **21**, wherein the at least one filter is a Boolean expression comprising two or more machines.

23. The system recited in claim **22**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

24. The system recited in claim **18**, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and wherein the at least one filter is directed to at least one of the processes.

25. The system recited in claim **24**, wherein the at least one filter is a Boolean expression comprising two or more processes.

26. The system recited in claim **25**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

27. The system recited in claim **18**, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and wherein the at least one filter is directed to at least one of the machines and at least one of the processes.

28. The system recited in claim **27**, wherein the at least one filter is a Boolean expression comprising two or more processes.

29. The system recited in claim **28**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

30. The system recited in claim **18**, wherein the at least one filter designates one or more machines, processes, in-process event creators, dynamic event creators, events, or threads.

31. The system recited in claim 18, wherein the at least one filter is a Boolean expression.

32. The system recited in claim 31, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

33. The system recited in claim 18, wherein the at least one filter is sent to one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

34. The system recited in claim 18, wherein the at least one filter is broadcast throughout the data processing system, wherein it is applied by one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

35. A method for analyzing the performance of a data processing system that produces events and that comprises a control station that analyzes events, and an event concentrator, the method comprising the steps of:

the control station specifying at least one filter; and

the event concentrator collecting events in accordance with the filter.

36. The method recited in claim 35, in which in the specifying step the at least one filter is a Boolean expression comprising two or more events.

37. The method recited in claim 36, in which in the specifying step the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

38. The method recited in claim 35, wherein the data processing system further comprises a plurality of machines, and further comprising the step of:

directing the at least one filter to at least one of the machines.

39. The method recited in claim 38, wherein the at least one filter is a Boolean expression comprising two or more machines.

40. The method recited in claim 39, in which in the specifying step the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

41. The method recited in claim 35, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and further comprising the step of:

directing the at least one filter to at least one of the processes.

42. The method recited in claim 41, wherein the at least one filter is a Boolean expression comprising two or more processes.

43. The method recited in claim 42, in which in the specifying step the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

44. The method recited in claim 35, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and further comprising the step of

directing the at least one filter to at least one of the machines and at least one of the processes.

45. The method recited in claim 44, wherein the at least one filter is a Boolean expression comprising two or more processes.

46. The method recited in claim 45, in which in the specifying step the Boolean expression comprises a set of variables, and farther comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

47. The method recited in claim 35, wherein the at least one filter designates one or more machines, processes, in-process event creators, dynamic event creators, events, or threads.

48. The method recited in claim 35, wherein the at least one filter is a Boolean expression.

49. The method recited in claim 48, in which in the specifying step the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

50. The method recited in claim 35, and further comprising the step of:

sending the at least one filter to one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

51. The method recited in claim 35, and further comprising the step of:

broadcasting the at least one filter throughout the data processing system, wherein it is applied by one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

52. The method recited in claim 35, wherein the steps recited therein can be performed in any suitable order.

53. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 35.

54. A method for analyzing the performance of a data processing system that comprises an in-process event creator that collects events generated by a first data source within the data processing system, a dynamic event creator that collects events that are generated on a time basis by a second data source within the data processing system, an event concentrator, and a control station that analyzes events, the method comprising the steps of:

the control station specifying at least one filter;

the in-process event creator and the dynamic event creator each collecting events in accordance with the at least one filter; and

the event concentrator collecting events from the in-process event creator and from the dynamic event creator in accordance with the at least one filter, and sending the events to the control station.

55. The method recited in claim 54, in which in the specifying step the at least one filter is a Boolean expression comprising two or more events.

56. The method recited in claim 55, in which in the specifying step the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

57. The method recited in claim 54, wherein the data processing system further comprises a plurality of machines, and further comprising the step of:

directing the at least one filter to at least one of the machines.

58. The method recited in claim 57, in which in the specifying step the at least one filter is a Boolean expression comprising two or more machines.

59. The method recited in claim 58, in which in the specifying step the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

60. The method recited in claim 54, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and further comprising the step of:

directing the at least one filter to at least one of the processes.

61. The method recited in claim 60, in which in the specifying step the at least one filter is a Boolean expression comprising two or more processes.

62. The method recited in claim 61, in which in the specifying step the Boolean expression comprises a sct of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

63. The method recited in claim 54, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and further comprising the step of

directing the at least one filter to at least one of the machines and at least one of the processes.

64. The method recited in claim 63, in which in the specifying step the at least one filter is a Boolean expression comprising two or more processes.

65. The method recited in claim 64, in which in the specifying step the Boolean expression comprises a sct of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

66. The method recited in claim 54, in which in the specifying step the at least one filter designates one or more machines, processes, in-process event creators, dynamic event creators, events, or threads.

67. The method recited in claim 54, in which in the specifying step the at least one filter is a Boolean expression.

68. The method recited in claim 67, in which in the specifying step the Boolean expression comprises a sct of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

69. The method recited in claim 54, and further comprising the step of:

sending the at least one filter to one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

70. The method recited in claim 54, and further comprising the step of:

broadcasting the at least one filter throughout the data processing system, wherein it is applied by one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

71. The method recited in claim 54, wherein the steps recited therein can be performed in any suitable order.

72. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 54.

73. In a system for analyzing the performance of a data processing system which produces events, and which has a graphical user interface including a display and a user interface selection device, a method of providing and selecting a filter from a menu on the display, the filter specifying which events to monitor, the method comprising the steps of:

displaying a menu on the display listing one or more items from a group comprising items representing event-generating machines, items representing event-generating components, and, items representing categories of events within the data processing system;

receiving a menu entry selection signal indicative of the user interface selection device selecting one of the items to monitor; and

repeating the immediately previous step, as necessary, until all desired items have been selected.

74. The method recited in claim 73, in which the at least one filter is a Boolean expression comprising two or more events.

75. The method recited in claim 74, in which the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

76. The method recited in claim 73, wherein the data processing system further comprises a plurality of machines, and further comprising the step of:

directing the at least one filter to at least one of the machines.

77. The method recited in claim 76, wherein the at least one filter is a Boolean expression comprising two or more machines.

78. The method recited in claim 77, in which the Boolean expression compriscs a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

79. The method recited in claim 73, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and further comprising the step of:

directing the at least one filter to at least one of the processes.

80. The method recited in claim 79, wherein the at least one filter is a Boolean expression comprising two or more processes.

81. The method recited in claim 80, in which the Boolean expression compriscs a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

82. The method recited in claim 73, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and further comprising the step of

directing the at least one filter to at least one of the machines and at least one of the processes.

83. The method recited in claim 82, wherein the at least one filter is a Boolean expression comprising two or more processes.

84. The method recited in claim 83, in which the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

85. The method recited in claim 73, wherein the at least one filter designates one or more machines, processes, in-process event creators, dynamic event creators, events, or threads.

86. The method recited in claim 73, wherein the at least one filter is a Boolean expression.

87. The method recited in claim 86, in which the Boolean expression compriscs a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

88. The method recited in claim 73, and further comprising the step of:

sending the at least one filter to one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

89. The method recited in claim 73, and further comprising the step of:

broadcasting the at least one filter throughout the data processing system, wherein it is applied by one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

90. The method recited in claim 73, wherein the steps recited therein can be performed in any suitable order.

91. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 73.

Filter Combination

1. A system for analyzing the performance of a data processing system that produces events, the system comprising:

at least one control station that analyzes events, the at least one control station specifying more than one filter to monitor events, or threads; and

an event concentrator, coupled to the at least one control station, that combines the filters and collects events in accordance with the filters.

2. The system recited in claim **1**, wherein at least one of the filters is a Boolean expression comprising two or more events.

3. The system recited in claim **2**, wherein the at least one Boolean expression comprises a set of variables and the at least one Boolean expression is modified by binding a subset of the variables.

4. The system recited in claim **1**, wherein the data processing system further comprises a plurality of machines, and wherein at least one filter is directed to at least one of the machines.

5. The system recited in claim **4**, wherein the at least one filter is a Boolean expression comprising two or more machines.

6. The system recited in claim **5**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

7. The system recited in claim **1**, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and wherein at least one filter is directed to at least one of the processes.

8. The system recited in claim **7**, wherein the at least one filter is a Boolean expression comprising two or more processes.

9. The system recited in claim **8**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

10. The system recited in claim **1**, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and wherein at least one filter is directed to at least one of the machines and at least one of the processes.

11. The system recited in claim **10**, wherein the at least one filter is a Boolean expression comprising two or more processes.

12. The system recited in claim **11**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

13. The system recited in claim **1**, wherein at least one filter designates one or more machines, processes, in-process event creators, dynamic event creators, events, or threads.

14. The system recited in claim **1**, wherein at least one filter is a Boolean expression.

15. The system recited in claim **14**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

16. The system recited in claim **1**, wherein at least one filter is sent to one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

17. The system recited in claim **1**, wherein at least one filter is broadcast throughout the data processing system, wherein it is applied by one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

18. The system recited in claim **1** and comprising two or more control stations, and wherein the event concentrator collects events from a plurality of sources within the data processing system and routes them to the respective control stations which specified that the events be monitored.

19. A system for analyzing the performance of a data processing system that produces events, the system comprising:

at least one control station that analyzes events, the at least one control station specifying more than one filter;

an in-process event creator that collects events generated by a first data source within the data processing system;

a dynamic event creator that collects events that are generated on a time basis by a second data source within the data processing system; and

an event concentrator collecting events from the in-process event creator and from the dynamic event creator, in accordance with a combination of the filters, and sending the events to the control station.

20. The system recited in claim **19**, wherein at least one of the filters is a Boolean expression comprising two or more events.

21. The system recited in claim **20**, wherein the at least one Boolean expression comprises a set of variables and the at least one Boolean expression is modified by binding a subset of the variables.

22. The system recited in claim **19**, wherein the data processing system further comprises a plurality of machines, and wherein at least one filter is directed to at least one of the machines.

23. The system recited in claim **22**, wherein the at least one filter is a Boolean expression comprising two or more machines.

24. The system recited in claim **23**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

25. The system recited in claim **19**, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and wherein at least one filter is directed to at least one of the processes.

26. The system recited in claim **25**, wherein the at least one filter is a Boolean expression comprising two or more processes.

27. The system recited in claim **26**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

28. The system recited in claim **19**, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and wherein at least one filter is directed to at least one of the machines and at least one of the processes.

29. The system recited in claim **28**, wherein the at least one filter is a Boolean expression comprising two or more processes.

30. The system recited in claim **29**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

31. The system recited in claim **19**, wherein at least one filter designates one or more machines, processes, in-process event creators, dynamic event creators, events, or threads.

32. The system recited in claim **19**, wherein at least one filter is a Boolean expression.

33. The system recited in claim **32**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

34. The system recited in claim **19**, wherein at least one filter is sent to one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

35. The system recited in claim **19**, wherein at least one filter is broadcast throughout the data processing system, wherein it is applied by one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

36. The system recited in claim **19** and comprising two or more control stations, and wherein the event concentrator collects events from a plurality of sources within the data processing system and routes them to the respective control stations which specified that the events be monitored.

37. A method for analyzing the performance of a data processing system that produces events and that comprises at least one control station that analyzes events, and an event concentrator, the method comprising the steps of:

the at least one control station specifying more than one filter; and

the event concentrator combining the filters and collecting events in accordance with the combined filter.

38. The method recited in claim **37**, in which at least one filter is a Boolean expression comprising two or more events.

39. The method recited in claim **38**, in which the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

40. The method recited in claim **37**, wherein the data processing system further comprises a plurality of machines, and further comprising the step of:

directing at least one filter to at least one of the machines.

41. The method recited in claim **40**, wherein the at least one filter is a Boolean expression comprising two or more machines.

42. The method recited in claim **41**, in which the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

43. The method recited in claim **37**, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and further comprising the step of:

directing at least one filter to at least one of the processes.

44. The method recited in claim **43**, wherein the at least one filter is a Boolean expression comprising two or more processes.

45. The method recited in claim **44**, in which the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

46. The method recited in claim **37**, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and further comprising the step of

directing at least one filter to at least one of the machines and at least one of the processes.

47. The method recited in claim **46**, wherein the at least one filter is a Boolean expression comprising two or more processes.

48. The method recited in claim **47**, in which the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

49. The method recited in claim **37**, wherein at least one filter designates one or more machines, processes, in-process event creators, dynamic event creators, events, or threads.

50. The method recited in claim **37**, wherein at least one filter is a Boolean expression.

51. The method recited in claim **50**, in which the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

52. The method recited in claim **37**, and further comprising the step of:

sending at least one filter to one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

53. The method recited in claim **37**, and further comprising the step of:

broadcasting at least one filter throughout the data processing system, wherein it is applied by one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

54. The method recited in claim **37**, wherein the data processing system comprises two or more control stations each specifying more than one filter, and further comprising the steps of:

the event concentrator collecting events from a plurality of sources within the data processing system; and

the event concentrator routing the events to the respective control stations which specified that the events be monitored.

55. The method recited in claim **37**, wherein the steps recited therein can be performed in any suitable order.

56. A computer-readable medium having computer-executable instructions for performing the steps recited in claim **37**.

57. A method for analyzing the performance of a data processing system that comprises an in-process event creator that collects events generated by a first data source within the data processing system, a dynamic event creator that collects events that are generated on a time basis

by a second data source within the data processing system, an event concentrator, and at least one control station that analyzes events, the method comprising the steps of:

the at least one control station specifying more than one filter;

the event concentrator combining the filters into a combined filter;

the in-process event creator and the dynamic event creator each collecting events in accordance with the combined filter; and

the event concentrator collecting events from the in-process event creator and from the dynamic event creator, in accordance with the combined filter, and sending the events to the at least one control station.

58. The method recited in claim 57, in which at least one filter is a Boolean expression comprising two or more events.

59. The method recited in claim 58, in which the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

60. The method recited in claim 57, wherein the data processing system further comprises a plurality of machines, and further comprising the step of:

directing at least one filter to at least one of the machines.

61. The method recited in claim 60, wherein the at least one filter is a Boolean expression comprising two or more machines.

62. The method recited in claim 61, in which the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

63. The method recited in claim 57, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and further comprising the step of:

directing at least one filter to at least one of the processes.

64. The method recited in claim 63, wherein the at least one filter is a Boolean expression comprising two or more processes.

65. The method recited in claim 64, in which the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

66. The method recited in claim 57, wherein the data processing system further comprises a plurality of machines, each running a plurality of processes, and further comprising the step of

directing at least one filter to at least one of the machines and at least one of the processes.

67. The method recited in claim 66, wherein the at least one filter is a Boolean expression comprising two or more processes.

68. The method recited in claim 67, in which the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

69. The method recited in claim 57, wherein at least one filter designates one or more machines, processes, in-process event creators, dynamic event creators, events, or threads.

70. The method recited in claim 57, wherein at least one filter is a Boolean expression.

71. The method recited in claim 70, in which the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

72. The method recited in claim 57, and further comprising the step of:

sending at least one filter to one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

73. The method recited in claim 57, and further comprising the step of:

broadcasting at least one filter throughout the data processing system, wherein it is applied by one or more specific machines, processes, in-process event creators, dynamic event creators, events, or threads.

74. The method recited in claim 57, wherein the data processing system comprises two or more control stations each specifying more than one filter, and further comprising the steps of:

the event concentrator collecting events from a plurality of sources within the data processing system; and

the event concentrator routing the events to the respective control stations which specified that the events be monitored.

75. The method recited in claim 57, wherein the steps recited therein can be performed in any suitable order.

76. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 57.

Filter Specification

1. A system for analyzing the performance of a data processing system that produces events, the system comprising:

a control station that analyzes events, the control station providing a graphical user interface for enabling a user to specify at least one filter; and

an event concentrator, coupled to the control station, that collects events in accordance with the at least one filter.

2. The system recited in claim 1, wherein the at least one filter is a Boolean expression comprising two or more events.

3. The system recited in claim 2, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

4. The system recited in claim 1, wherein the graphical user interface comprises a window for displaying a textual representation of the at least one filter.

5. The system recited in claim 4, wherein the at least one filter can be entered by the user as text.

6. The system recited in claim 1, wherein the control station analyzes events collected by the event concentrator as the events are collected.

7. The system recited in claim 1, wherein the control station analyzes events collected by the event concentrator after the events are collected.

8. The system recited in claim 1, wherein the graphical user interface comprises a pre-defined list of filters from which a user can specify at least one filter.

9. The system recited in claim 1, wherein the at least one filter is a debug or trace switch.

10. A system for analyzing the performance of a data processing system that produces events, the system comprising:

a control station that analyzes events, the control station providing a graphical user interface for enabling a user to specify at least one filter;

an in-process event creator that collects events generated by a first data source within the data processing system;

a dynamic event creator that collects events that are generated on a time basis by a second data source within the data processing system; and

an event concentrator collecting events from the in-process event creator and from the dynamic event creator, in accordance with the at least one filter, and sending the events to the control station.

11. The system recited in claim **10**, wherein the at least one filter is a Boolean expression comprising two or more events.

12. The system recited in claim **11**, wherein the Boolean expression comprises a set of variables and the Boolean expression is modified by binding a subset of the variables.

13. The system recited in claim **10**, wherein the graphical user interface comprises a window for displaying a textual representation of the at least one filter.

14. The system recited in claim **13**, wherein the at least one filter can be entered by the user as text.

15. The system recited in claim **10**, wherein the control station analyzes events collected by the event concentrator as the events are collected.

16. The system recited in claim **10**, wherein the control station analyzes events collected by the event concentrator after the events are collected.

17. The system recited in claim **10**, wherein the graphical user interface comprises a pre-defined list of filters from which a user can specify at least one filter.

18. The system recited in claim **10**, wherein the at least one filter is a debug or trace switch.

19. A method for analyzing the performance of a data processing system that produces events and that comprises a control station that analyzes events, and an event concentrator, the method comprising the steps of:

the control station providing a graphical user interface for enabling a user to specify at least one filter; and

the event concentrator collecting events in accordance with the filter.

20. The method recited in claim **19**, wherein the at least one filter is a Boolean expression comprising two or more events.

21. The method recited in claim **20**, wherein the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

22. The method recited in claim **19**, wherein the graphical user interface comprises a window, and further comprising the step of:

displaying a textual representation of the at least one filter in the window.

23. The method recited in claim **19**, and further comprising the step of:

providing a window as part of the graphical user interface, into which window the at least one filter can be entered by the user as text.

24. The method recited in claim **19**, wherein the control station analyzes events collected by the event concentrator as the events are collected.

25. The method recited in claim **19**, wherein the control station analyzes events collected by the event concentrator after the events are collected.

26. The method recited in claim **19**, wherein the graphical user interface comprises a pre-defined list of filters from which a user can specify at least one filter.

27. The method recited in claim **19**, wherein the at least one filter is a debug or trace switch.

28. The method recited in claim **19**, wherein the steps recited therein can be performed in any suitable order.

29. A computer-readable medium having computer-executable instructions for performing the steps recited in claim **19**.

30. A method for analyzing the performance of a data processing system that comprises an in-process event creator that collects events generated by a first data source within the data processing system, a dynamic event creator that collects events that are generated on a time basis by a second data source within the data processing system, an event concentrator, and a control station that analyzes events, the method comprising the steps of:

the control station providing a graphical user interface for enabling a user to specify at least one filter;

the in-process event creator and the dynamic event creator each collecting events in accordance with the at least one filter; and

the event concentrator collecting events from the in-process event creator and from the dynamic event creator, in accordance with the at least one filter, and sending the events to the control station.

31. The method recited in claim **30**, wherein the at least one filter is a Boolean expression comprising two or more events.

32. The method recited in claim **31**, wherein the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

33. The method recited in claim **30**, wherein the graphical user interface comprises a window, and further comprising the step of:

displaying a textual representation of the at least one filter in the window.

34. The method recited in claim **30**, and further comprising the step of:

providing a window as part of the graphical user interface, into which window the at least one filter can be entered as text.

35. The method recited in claim **30**, wherein the control station analyzes events collected by the event concentrator as the events are collected.

36. The method recited in claim **30**, wherein the control station analyzes events collected by the event concentrator after the events are collected.

37. The method recited in claim **30**, wherein the graphical user interface comprises a pre-defined list of filters from which a user can specify at least one filter.

38. The method recited in claim **30**, wherein the at least one filter is a debug or trace switch.

39. The method recited in claim **30**, wherein the steps recited therein can be performed in any suitable order.

40. A computer-readable medium having computer-executable instructions for performing the steps recited in claim **30**.

41. In a system for analyzing the performance of a data processing system which produces events, and which has a graphical user interface including a display and a user interface selection device, a method of providing and selecting a filter from a menu on the display, the filter

specifying which events to monitor, the method comprising the steps of:

displaying a menu on the display listing items representing event-generating machines, event-generating components, and categories of events within the data processing system;

receiving a menu entry selection signal indicative of the user interface selection device selecting one of the items to monitor; and

repeating the immediately previous step, as necessary, until all desired items have been selected.

42. The method recited in claim 41, wherein the at least one filter is a Boolean expression comprising two or more events.

43. The method recited in claim 42, wherein the Boolean expression comprises a set of variables, and further comprising the step of:

modifying the Boolean expression by binding a subset of the variables.

44. The method recited in claim 41, wherein the graphical user interface comprises a window, and further comprising the step of:

displaying a textual representation of the at least one filter in the window.

45. The method recited in claim 41, and further comprising the step of:

providing a window as part of the graphical user interface, into which window the at least one filter can be entered as text.

46. The method recited in claim 41, wherein the control station analyzes events collected by the event concentrator as the events are collected.

47. The method recited in claim 41, wherein the control station analyzes events collected by the event concentrator after the events are collected.

48. The method recited in claim 41, wherein the graphical user interface comprises a pre-defined list of filters from which a user can specify at least one filter.

49. The method recited in claim 41, wherein the items representing event-generating components refer to data sources on event-generating machines being monitored.

50. The method recited in claim 41, wherein the at least one filter is a debug or trace switch.

51. The method recited in claim 41, wherein the steps recited therein can be performed in any suitable order.

52. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 41.

APIs

1. A computer system comprising:

a computer comprising a processor and a memory operatively coupled together;

an operating system executing in the processor;

an application program running under the control of the operating system, the application program having an event-generating component; and

application program interfaces associated with the event-generating component operative to receive data from the operating system and send data to the operating system.

2. The computer system of claim 1, wherein the application program interfaces comprise:

a first interface that enables the operating system to set or disable a status condition in the application; and

a second interface that receives a status query from the operating system and that returns the status of the status condition to the operating system.

3. The computer system of claim 1, wherein the application program interfaces comprise:

an interface that enables the operating system to read any one or more of several fields in the application, the fields being from a group comprising arguments, causality i.d., correlation i.d., dynamic event data, exception, return value, security i.d., source component, source handle, source machine, source process, source process name, source session, source thread, target component, target handle, target machine, target process, target process name, target session, and target thread.

4. A computer system comprising:

a computer comprising a processor and a memory operatively coupled together;

an operating system executing in the processor, the operating system having an event-registering component;

an application program running under the control of the operating system; and

application program interfaces associated with the event-registering component operative to receive data from the operating system and send data to the operating system.

5. The computer system of claim 4, wherein the application program interfaces comprise:

a first interface that enables the operating system to query whether a status condition is set or disabled in the application; and

a second interface that returns data to the operating system only if the status condition is set.

6. The computer system of claim 4, wherein the application program interfaces comprise:

an interface that enables the operating system to read any one or more of several fields in the application, the fields being from a group comprising arguments, causality i.d., correlation i.d., dynamic event data, exception, return value, security i.d., source component, source handle, source machine, source process, source process name, source session, source thread, target component, target handle, target machine, target process, target process name, target session, and target thread.

7. A set of application program interfaces embodied on a computer-readable medium for execution on a computer in conjunction with an operating system that interfaces with an application program having an event-generating component, comprising:

a first interface that enables the operating system to set or disable a status condition in the application; and

a second interface that receives a status query from the operating system and that returns the status condition to the operating system.

8. The set of application program interfaces recited in claim 7 and further comprising:

a third interface that enables the operating system to read any one or more of several fields in the application, the fields being from a group comprising arguments, causality i.d., correlation i.d., dynamic event data, exception, return value, security i.d., source component, source handle, source machine, source process, source process name, source session, source thread, target component, target handle, target machine, target process, target process name, target session, and target thread.

9. A set of application program interfaces embodied on a computer-readable medium for execution on a computer

in conjunction with an application program that interfaces with an operating system having an event-registering component, comprising:

a first interface that enables the operating system to query whether a status condition is set or disabled in the application; and

a second interface that returns data to the operating system only if the status condition is set.

10. The set of application program interfaces recited in claim 9 and further comprising:

a third interface that enables the operating system to read any one or more of several fields in the application, the fields being from a group comprising arguments, causality i.d., correlation i.d., dynamic event data, exception, return value, security i.d., source component, source handle, source machine, source process, source process name, source session, source thread, target component, target handle, target machine, target process, target process name, target session, and target thread.

Animated Application Model

1. A system for analyzing the performance of a data processing system that produces events, the system comprising:

an event concentrator that collects events; and

a control station, coupled to the event concentrator, that analyzes events, the control station displaying a model of the functionally active structure of the data processing system.

2. The system recited in claim **1**, wherein the model is a hierarchical model.

3. The system recited in claim **2**, wherein the hierarchical model includes items from a group comprising machines, processes, data sources, entities, and instances.

4. The system recited in claim **1**, wherein the control station displays items from a group comprising summary data, time data, event details, and a call tree simultaneously while displaying the model.

5. The system recited in claim **1**, wherein the model is updated in real time as the control station analyzes the events.

6. The system recited in claim **1**, wherein the control station comprises a graphical user interface including a display and a user interface selection device.

7. The system recited in claim **6**, wherein the graphical user interface employs a video cassette recorder (VCR) paradigm to enable a user to analyze the performance of the data processing system.

8. The system recited in claim **7**, wherein the VCR paradigm displays user-selectable commands from a group comprising playing, replaying, stopping, reversing, pausing, and changing the speed of the model.

9. The system recited in claim **1**, wherein the control station uses heuristics from a group comprising time-ordering, causality information, and event handles to display the model.

10. The system recited in claim **1**, wherein the control station displays active portions of the model in a visually distinctive manner.

11. A system for analyzing the performance of a data processing system that produces events, the system comprising:

an in-process event creator that collects events generated by a first data source within the data processing system;

a dynamic event creator that collects events that are generated on a time basis by a second data source within the data processing system;

an event concentrator collecting events from the in-process event creator and from the dynamic event creator; and

a control station, coupled to the event concentrator, that analyzes events, the control station displaying a model of the functionally active structure of the data processing system.

12. The system recited in claim **11**, wherein the model is a hierarchical model.

13. The system recited in claim **12**, wherein the hierarchical model includes items from a group comprising machines, processes, data sources, entities, and instances.

14. The system recited in claim **11**, wherein the control station displays items from a group comprising summary data, time data, event details, and a call tree simultaneously while displaying the model.

15. The system recited in claim **11**, wherein the model is updated in real time as the control station analyzes the events.

16. The system recited in claim **11**, wherein the control station comprises a graphical user interface including a display and a user interface selection device.

17. The system recited in claim **16**, wherein the graphical user interface employs a video cassette recorder (VCR) paradigm to enable a user to analyze the performance of the data processing system.

18. The system recited in claim **17**, wherein the VCR paradigm displays user-selectable commands from a group comprising playing, replaying, stopping, reversing, pausing, and changing the speed of the model.

19. The system recited in claim **11**, wherein the control station uses heuristics from a group comprising time-ordering, causality information, and event handles to display the model.

20. The system recited in claim **11**, wherein the control station displays active portions of the model in a visually distinctive manner.

21. A method for analyzing and displaying the performance of a data processing system that produces events and that comprises a control station that analyzes events, and an event concentrator, the method comprising the steps of:

the event concentrator collecting events; and

the control station displaying a model of the functionally active structure of the data processing system.

22. The method recited in claim **21**, wherein the model is a hierarchical model.

23. The method recited in claim 22, wherein the hierarchical model includes items from a group comprising machines, processes, data sources, entities, and instances.

24. The method recited in claim **21**, and further comprising the step of:

the control station displaying items from a group comprising summary data, time data, event details, and a call tree simultaneously while displaying the model.

25. The method recited in claim 21, and further comprising the step of:

updating the model in real time as the control station analyzes the events.

26. The method recited in claim **21**, wherein the control station comprises a graphical user interface including a display and a user interface selection device.

27. The method recited in claim **26**, wherein the graphical user interface employs a video cassette recorder (VCR) paradigm to enable a user to analyze the performance of the data processing system.

28. The method recited in claim **27**, and further comprising the step of:

the VCR paradigm displaying user-selectable commands from a group comprising playing, replaying, stopping, reversing, pausing, and changing the speed of the model.

29. The method recited in claim 21, and further comprising the step of:

the control station using heuristics from a group comprising time-ordering, causality information, and event handles to generate and display the model.

30. The method recited in claim 21, and further comprising the step of:

the control station displaying active portions of the model in a visually distinctive manner.

31. The method recited in claim 21, wherein the steps recited therein can be performed in any suitable order.

32. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 21.

33. A method for analyzing and displaying the performance of a data processing system that comprises an in-process event creator that collects events generated by a first data source within the data processing system, a dynamic event creator that collects events that are generated on a time basis by a second data source within the data processing system, an event concentrator, and a control station that analyzes events, the method comprising the steps of:

the in-process event creator and the dynamic event creator each collecting events from their respective data sources;

the event concentrator collecting events from the in-process event creator and from the dynamic event creator and sending the events to the control station; and

the control station displaying a model of the functionally active structure of the data processing system.

34. The method recited in claim 33, wherein the control station provides a user interface for enabling user selection of one or more portions of the model.

35. The method recited in claim 34, and further comprising the step of:

exploding a portion of the model to show more detail, in response to a user selection.

36. The method recited in claim 34, and further comprising the step of:

contracting a portion of the model to show less detail, in response to a user selection.

37. The method recited in claim 33, wherein the model is a hierarchical model.

38. The method recited in claim 37, wherein the hierarchical model includes items from a group comprising machines, processes, data sources, entities, and instances.

39. The method recited in claim 33, and further comprising the step of:

the control station displaying items from a group comprising summary data, time data, event details, and a call tree simultaneously while displaying the model.

40. The method recited in claim 33, and further comprising the step of:

updating the model in real time as the control station analyzes the events.

41. The method recited in claim 33, wherein the control station comprises a graphical user interface including a display and a user interface selection device.

42. The method recited in claim 41, wherein the graphical user interface employs a video cassette recorder (VCR) paradigm to enable a user to analyze the performance of the data processing system.

43. The method recited in claim 42, and further comprising the step of:

the VCR paradigm displaying user-selectable commands from a group comprising playing, replaying, stopping, reversing, pausing, and changing the speed of the model.

44. The method recited in claim 33, and further comprising the step of:

the control station using heuristics from a group comprising time-ordering, causality information, and event handles to generate and display the model.

45. The method recited in claim 33, and further comprising the step of:

the control station displaying active portions of the model in a visually distinctive manner.

46. The method recited in claim 33, wherein the steps recited therein can be performed in any suitable order.

47. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 33.

48. In a system for analyzing the performance of a data processing system which produces events, and which has a graphical user interface including a display and a user interface selection device, a method of providing an animated model of the performance of the data processing system and enabling user expansion or contraction of portions of the animated model, the method comprising the steps of:

displaying the model showing the functionally active structure of the data processing system;

receiving a selection signal indicative of the user interface selection device selecting a portion of the animated model;

receiving an expansion or contraction command; and

performing an expansion or contraction function on the selected portion in response to the selection signal and the expansion or contraction command, as appropriate.

49. The method recited in claim 48, wherein the model is a hierarchical model.

50. The method recited in claim 49, wherein the hierarchical model includes items from a group comprising machines, processes, data sources, entities, and instances.

51. The method recited in claim 48, and further comprising the step of:

the control station displaying items from a group comprising summary data, time data, event details, and a call tree simultaneously while displaying the model.

52. The method recited in claim 48, and further comprising the step of:

updating the model in real time as the control station analyzes the events.

53. The method recited in claim 48, wherein the graphical user interface employs a video cassette recorder (VCR) paradigm to enable a user to analyze the performance of the data processing system.

54. The method recited in claim 53, and further comprising the step of:

the VCR paradigm displaying user-selectable commands from a group comprising playing, replaying, stopping, reversing, pausing, and changing the speed of the model.

55. The method recited in claim 48, and further comprising the step of:

the control station using heuristics from a group comprising time-ordering, causality information, and event handles to generate and display the model.

56. The method recited in claim 48, and further comprising the step of:

the control station displaying active portions of the model in a visually distinctive manner.

57. The method recited in claim 48, wherein the steps recited therein can be performed in any suitable order.

58. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 48.

Performance Analysis

1. A system for analyzing the performance of a data processing system that produces events, the system comprising:

an event concentrator that collects events; and

a control station, coupled to the event concentrator, that analyzes events, the control station displaying a call tree of the functionally active structure of the data processing system.

2. The system recited in claim 1, wherein the control station displays items from a group comprising Gantt style charts, process summary data, performance summary data, and time data simultaneously while displaying the call tree.

3. The system recited in claim 2, wherein the displayed items are time-synchronized.

4. The system recited in claim 1, wherein the call tree is updated in real time as the control station analyzes the events.

5. The system recited in claim 1, wherein the control station analyzes events collected by the event concentrator as the events are collected.

6. The system recited in claim 1, wherein the control station analyzes events collected by the event concentrator after the events are collected.

7. The system recited in claim 1, wherein the control station provides a user interface for enabling user selection of one or more portions of the call tree.

8. The system recited in claim 7, wherein the control station explodes a portion of the call tree to show more detail, in response to a user selection.

9. The system recited in claim 7, wherein the control station contracts a portion of the call tree to show less detail, in response to a user selection.

10. The system recited in claim 1, wherein the control station uses heuristics from a group comprising time-ordering, causality information, and event handles to display the call tree.

11. A system for analyzing the performance of a data processing system that produces events, the system comprising:

an in-process event creator that collects events generated by a first data source within the data processing system;

a dynamic event creator that collects events that are generated on a time basis by a second data source within the data processing system;

an event concentrator collecting events from the in-process event creator and from the dynamic event creator, and sending the events to the control station; and

a control station that analyzes events, the control station displaying a call tree of the functionally active structure of the data processing system.

12. The system recited in claim 12, wherein the control station displays items from a group comprising Gantt style charts, process summary data, performance summary data, and time data simultaneously while displaying the call tree.

13. The system recited in claim 12, wherein the displayed items are time-synchronized.

14. The system recited in claim 11, wherein the call tree is updated in real time as the control station analyzes the events.

15. The system recited in claim 11, wherein the control station analyzes events collected by the event concentrator as the events are collected.

16. The system recited in claim 11, wherein the control station analyzes events collected by the event concentrator after the events are collected.

17. The system recited in claim 11, wherein the control station provides a user interface for enabling user selection of one or more portions of the call tree.

18. The system recited in claim 17, wherein the control station explodes a portion of the call tree to show more detail, in response to a user selection.

19. The system recited in claim 17, wherein the control station contracts a portion of the call tree to show less detail, in response to a user selection.

20. The system recited in claim 11, wherein the control station uses heuristics from a group comprising time-ordering, causality information, and event handles to display the call tree.

21. A method for analyzing and displaying the performance of a data processing system that produces events and that comprises a control station that analyzes events, and an event concentrator, the method comprising the steps of:

the event concentrator collecting events; and

the control station displaying a call tree of the functionally active structure of the data processing system.

22. The method recited in claim 21, and further comprising the step of:

the control station displaying items from a group comprising Gantt style charts, process summary data, performance summary data, and time data simultaneously while displaying the call tree.

23. The method recited in claim 22, wherein the displayed items are time-synchronized.

24. The method recited in claim 21, and further comprising the step of:

updating the call tree in real time as the control station analyzes the events.

25. The method recited in claim 21, and further comprising the step of:

the control station analyzing events collected by the event concentrator as the events are collected.

26. The method recited in claim 21, and further comprising the step of:

the control station analyzing events collected by the event concentrator after the events are collected.

27. The method recited in claim 21, wherein the control station comprises a graphical user interface including a display and a user interface selection device.

28. The method recited in claim 27, and further comprising the step of:

exploding a portion of the call tree to show more detail, in response to a user selection.

29. The method recited in claim 27, and further comprising the step of:

contracting a portion of the call tree to show less detail, in response to a user selection.

30. The method recited in claim 21, and further comprising the step of:

the control station using heuristics from a group comprising time-ordering, causality information, and event haodles to generate and display the call tree.

31. The method recited in claim 21, wherein the steps recited therein cao he performed in any suitahle order.

32. A computer-readahle medium having computer-excutable instructions for performing the steps recited in claim 21.

33. A method for analyzing and displaying the performance of a data processing system that comprises an in-process event creator that collects events generated by a first data source within the data processing system, a dynamic event creator that collects events that are generated on a time basis by a second data source within the data processing system, an event concentrator, and a control station that analyzes events, the method comprising the steps of:

the in-process event creator and the dynamic event creator each collecting events from their respective data sources;

the event concentrator collecting events from the in-process event creator and from the dynamic event creator and sending the events to the control station; and

the control station displaying a call tree of the functionally active structure of the data processing system, and further providing a user interface for enabling user selection of one or more portions of the call tree.

34. The method recited in claim 33, and further comprising the step of:

the control station displaying items from a group comprising Gantt style charts, process summary data, performance summary data, and time data simultaneously while displaying the call tree.

35. The method recited in claim 34, wherein the displayed items are time-synchronized.

36. The method recited in claim 33, and further comprising the step of:

updating the call tree in real time as the control station analyzes the events.

37. The method recited in claim 33, and further comprising the step of:

the control station analyzing events collected by the event concentrator as the events are collected.

38. The method recited in claim 33, and further comprising the step of:

the control station analyzing events collected by the event concentrator after the events are collected.

39. The method recited in claim 33, wherein the control station comprises a graphical user interface including a display and a user interface selection device.

40. The method recited in claim 39, and further comprising the step of:

exploding a portion of the call tree to show more detail, in response to a user selection.

41. The method recited in claim 39, and further comprising the step of:

contracting a portion of the call tree to show less detail, in response to a user selection.

42. The method recited in claim 33, and further comprising the step of:

the control station using heuristics from a group comprisiog time-ordering, causality information, and event haodles to generate and display the call tree.

43. The method recited in claim 33, wherein the steps recited therein can be performed in any suitable order.

44. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 33.

45. In a system for analyzing the performance of a data processing system which produces events, and which has a graphical user interface including a display and a user interface selection device, a method of providing a call tree of the performance of the data processing system and enabling user expansion or coatraction of portions of the call tree, the method comprising the steps of:

displaying the call tree showing the functionally active structure of the data processing system;

receiving a selection signal indicative of the user interface selection device selecting a portion of the call tree;

receiving an expansion or contraction command; and

performing an expansion or contraction function on the selected portion in response to the selection signal and the expansion or contraction command, as appropriate.

46. The method recited in claim 45, and further comprising the step of:

the control station displaying items from a group comprising Gantt style charts, process summary data, performance summary data, and time data simultaneously while displaying the call tree.

47. The method recited in claim 46, wherein the displayed items are time-synchronized.

48. The method recited in claim 45, and further comprising the step of:

updating the call tree in real time as the control station analyzes the events.

49. The method recited in claim 45, and further comprising the step of:

the control station analyzing events collected by the event concentrator as the events are collected.

50. The method recited in claim 45, and further comprising the step of:

the control station analyzing events collected by the event concentrator after the events are collected.

51. The method recited in claim 45, and further comprising the step of:

the control station using heuristics from a group comprising time-ordering, causality information, and event handles to generate and display the call tree.

52. The method recited in claim 45, wherein the steps recited therein can be performed in any suitable order.

53. A computer-readable medium having computer-executable instructions for performing the steps recited in claim 45.

1. A system for analyzing the performance of a data processing system comprising:

a control station adapted to control at least one event concentrator to enable monitoring of a process;

an in-process event creator associated with the monitored process, the event creator collecting events geoerated hy the monitored process when enabled by the at least one event concentrator; and

the at least one event concentrator, coupled to the control station and to the in-process event creator, that collects events from the in-process event creator and sends them to the control station.

2. A system as recited in claim 1, wherein the in-process event creator is coupled to the control statioo and can be turned on or off by the control station.

3. The system recited in claim 1, wherein the io-process event creator can be created and removed by the control station.

4. A system as recited in claim 1, and further comprising:

a dynamic event creator, coupled to the event concentrator, that collects data that is generated on a time basis;

and wherein the event concentrator collects data from the dynamic event creator and sends it to the control station.

5. A system as recited in claim 4, wherein the dynamic event creator is coupled to the control station and can be turned on or off by the control station.

6. The system recited in claim 4, wherein the dynamic event creator can be created and removed by the control station.

7. A system for analyzing the structure and operation of an application executing on a data processing system comprising:

a control station adapted to control at least one event concentrator to enable monitoring of an application;

an in-process event creator associated with the monitored application, the event creator collecting events generated by the execution of the application when enabled by the at least one event concentrator; and

an event concentrator, coupled to the control station and to the in-process event creator, that collects events from the in-process event creator and sends them to the control station.

8. A system as recited in claim 7, wherein the in-process event creator is coupled to the control station and can be turned on or off by the control station.

9. The system recited in claim 7, wherein the in-process event creator can be created and removed by the control station.

10. A system as recited in claim 7, and further comprising:

a dynamic event creator, coupled to the event concentrator, that collects data that is generated on a time basis from the execution of the application;

and wherein the event concentrator collects data from the dynamic event creator and sends it to the control station.

11. A system as recited in claim 10, wherein the dynamic event creator is coupled to the control station and can be turned on or off by the control station.

12. The system recited in claim 10, wherein the dynamic event creator can be created and removed by the control station.

13. The system as recited in claim 7 wherein the application is executing on two or more data processing systems.

14. A system for analyzing the performance of a network comprising two or more data processing systems comprising:

a control station adapted to control at least one event concentrator to enable monitoring of a process;

an in-process event creator associated with the monitored process, the event creator collecting events generated hy the monitored process when enabled by the at least one event concentrator; and

an event concentrator, coupled to the control station and to the in-process event creator, that collects events from the in-process event creator and sends them to the control station.

15. A system as recited in claim 14, wherein the in-process event creator is coupled to the control station and can he turned on or off by the control station.

16. The system recited in claim 14, wherein the in-process event creator can be created and removed by the control station.

17. The system recited in claim 14, wherein a separate in-process event creator can be created and removed hy the control station for each data processing system.

18. A system as recited in claim 14, and further comprising:

a dynamic event creator, coupled to the event concentrator, that collects data that is generated on a time basis;

and wherein the event concentrator collects data from the dynamic event creator and sends it to the control station.

19. A system as recited in claim 18, wherein the dynamic event creator is coupled to the control station and can be turned on or off by the control station.

20. The system recited in claim 18, wherein the dynamic event creator can be created and removed by the control station.

21. A method of analyzing the performance of a data processing system having at least one program module, a control station, and an event concentrator, the method comprising the steps of:

the at least one program module creating an in-process event creator;

the in-process event creator collecting events generated by a data source within the data processing system; and

the event concentrator collecting events from the in-process event creator and sending them to the control station.

22. The method recited in claim 21, wherein the control station turns the in-process event creator on or off.

23. The method recited in claim 22, wherein the control station turns the in-process event creator on or off by setting or disabling a status condition in the in-process event creator.

24. The method recited in claim 21, wherein the event concentrator buffers a predetermined quantity of the events and only stores the events on request of the control station.

25. The method recited in claim 21, further comprising the steps of:

the at least one program module creating a dynamic event creator;

the dynamic event creator collecting data that is generated on a time basis; and

the event concentrator collecting data from the dynamic event creator and sending it to the control station.

26. The method recited in claim 25, wherein the control station turns the dynamic event creator on or off.

27. The method recited in claim 26, wherein the event concentrator buffers a predetermined quantity of the data and only stores the data on request of the control station.

28. The method recited in claim 21, wherein the at least one program module first creates an in-process event creator reference in the creating step, and further comprising the step of: a local event concentrator converting the in-process event creator reference to an in-process event creator.

29. The method recited in claim 21, wherein the control station creates the event concentrator.

30. The method recited in claim 21, wherein the steps recited therein can be performed in any suitable order.

31. A computer-readable medium having computer-executable instructions for analyzing the performance of a data processing system having at least one program module, a control station and an event concentrator, the computer-executable instructions performing the steps comprising:

the at least one program module creating an in-process event creator;

the in-process event creator collecting events generated by a data source within the data processing system; and

the event concentrator collecting events from the in-process event creator and sending them to the control station.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,467,052 B1
DATED : October 15, 2002
INVENTOR(S) : Kaler et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 28,
Line 58, please delete "C VSA Parameter Value DWORD= 0.3000" and insert therefor -- C VSA Parameter Value DWORD= 0.30000 --.

Signed and Sealed this

Twenty-ninth Day of April, 2003

JAMES E. ROGAN
Director of the United States Patent and Trademark Office